Maîtrise énergétique des centres de données virtualisés : D'un scénario de charge à l'optimisation du placement des calculs

Guillaume Le Louët

Thèse de Doctorat

Guillaume LE LOUËT

Mémoire présenté en vue de l'obtention du
grade de Docteur de l'École nationale supérieure des mines de Nantes
sous le label de l'Université de Nantes Angers Le Mans

École doctorale : Sciences et technologies de l'information, et mathématiques

Discipline : Informatique et applications, section CNU 27
Unité de recherche : Laboratoire d'informatique de Nantes-Atlantique (LINA)

Soutenue le 12 mai 2014
Thèse nř : 2014EMNA0119

Maîtrise énergétique des centres de données virtualisés

D'un scénario de charge à l'optimisation du placement des calculs

JURY

Rapporteurs :	**M. Noël DE PALMA**, Professeur, Université de Grenoble
	M. Daniel HAGIMONT, Professeur, ENSEEIHT de Toulouse
	M. Jean-Pierre GUÉDON, Professeur, Université de Nantes
Examinateurs :	**M^{me} Anne-Cécile ORGERIE**, Chargée de Recherche CNRS, Centre de recherche Inria Rennes
	M. Mario SÜDHOLT, Professeur, Mines de Nantes
Directeur de thèse :	**M. Jean-Marc MENAUD**, Professeur, Mines de Nantes

Remerciements

Cette thèse est l'aboutissement d'un travail de quatre ans, sous la direction de mon directeur Jean-Marc Menaud. Je tiens tout particulièrement à remercier celui-ci pour la patience, la confiance et la pédagogie dont il a fait preuve à mon égard et qui m'ont permis de mener à bien ce travail. Je le remercie bien évidemment de m'avoir offert cette possibilité, en me proposant sujet de thèse, encadrement et financement.

Outre mon directeur, mon jury de thèse m'a permis de valider cette dernière. Je remercie celui-ci, en commençant par mes rapporteurs, le professeur Noël De Palma de Grenoble, le professeur Daniel Hagimont de Toulouse, et le professeur Jean-Pierre Guédon de Nantes, qui ont relu ma thèse et m'ont permis de corriger mes erreurs. Je remercie bien évidemment les autres membres du jury, Mme Anne-Cécile Orgerie de Rennes et le professeur Mario Südholt de Nantes, pour leur déplacement et leur participation à ma soutenance.

Mon travail de thèse ne s'est pas fait sans mal, mais j'ai eu la chance de trouver de la patience et de la pédagogie parmi certains collègues. Outre mon directeur, je voudrais remercier Charles Prud'homme et Sophie Demassey pour leurs explications de la programmation par contraintes, domaine très compliqué et qui nécessite d'adapter sa méthode de raisonnement au problème et non l'inverse.

Cette programmation par contraintes a été utilisée par Fabien Hermenier pour le développement de Entropy, que je remercie pour le travail important qu'il a réalisé sur ce projet. Je le remercie aussi pour son ouverture d'esprit durant les quelques discussion que nous avons eues, qui m'ont permis de progresser dans mes travaux.

Concernant Stresscloud, je tiens à remercier mes collègues Fréderic Dumont et Alexandre Garnier pour leur enthousiasme, leur flegme et le temps que nous avons passé ensemble à discuter et développer, sur des domaines vastes tels les drivers, l'interaction de code dans différents langages, les capteurs, l'analyse de données.

Je remercie l'ensemble du département informatique qui en fait un lieu de travail très agréable, et plus particulièrement Hervé Graal et Jacques Noyé pour leur sensibilité et l'impression que j'avais de m'enrichir à leur contact. Bien sûr, la capacité de travail et cette ambiance ne sont possibles que parce que la logistique du département est assurée de main de maître, pour cela je tiens à remercier le secrétariat du département, et en particulier Florence Rogues qui m'a rattrapé quelques retards avec une bonne humeur

bien agréable.

Enfin, d'un point de vue plus personnel, je remercie ma grand mère pour tout ce qu'elle a fait, Dominique et Georges Valade, et mes oncles et tantes qui se sont occupés de moi dans mon enfance, m'ont apporté de la confiance en moi et fait découvrir l'informatique alors que j'étais insupportable.

Table des matières

1

Présentation de la thèse

Sommaire

L'objectif de ce chapitre est de présenter succinctement les travaux de cette thèse : son positionnement, la problématique traitée et les contributions effectuées. Il permet au lecteur d'avoir un résumé reprenant les éléments essentiels de cette thèse. Chacun d'eux sera présenté plus précisément dans le reste du manuscrit.

1.1 Les centres de données physiques et virtuels

Cette thèse se positionne dans le domaine du *cloud computing*. Le cloud computing est à la fois un nouveau modèle d'utilisation des ressources de calculs et un nouveau modèle économique pour les éditeurs logiciels. Les fondements du cloud computing sont : l'externalisation des applications et le paiement à la demande.

L'externalisation des applications impose un nouveau modèle technologique. L'application n'est plus exécutée sur la machine de l'utilisateur, mais hébergée sur un serveur distant. Il est donc nécessaire que le client dispose d'une connexion internet pour accéder à l'application. Dans ce modèle, l'éditeur logiciel ne se contente plus de simplement

vendre l'application, mais se doit également de l'héberger. Suivant le nombre de ses clients, l'éditeur logiciel se doit de maintenir une infrastructure matérielle permettant d'exécuter convenablement l'application. Cette infrastructure peut donc comporter de quelques serveurs à plusieurs milliers. Pour exemple, la Direction générale des Finances publiques propose depuis 2009 la déclaration en ligne des revenus des particuliers. Cette application nécessite l'utilisation de plus de 5000 serveurs pour absorber les 9 millions d'utilisateurs. Ces serveurs physiques sont hébergés dans des salles informatiques dédiées, nommées centres de données, ou sont contrôlés en permanence la température, l'hygrométrie, le niveau de poussière et le niveau sonore. En plus de ces contrôles permanents, le centre de données doit disposer d'équipement redondant pour assurer une haute disponibilité de son infrastructure (plusieurs alimentations, climatisation, réseau électrique etc.), ce qui entraîne des coûts de construction et de maintenance très importants. La construction d'une telle salle n'est justifiée que pour les éditeurs logiciels possédant plusieurs milliers de clients comme Google, Amazon, Facebook, Apple etc.

Dans la très grande majorité des cas, les éditeurs logiciels louent des ressources matérielles dans des centres d'hébergement. Avant l'arrivé du cloud computing, l'éditeur achetait ou louait des serveurs physiques dédiés installés physiquement dans un centre de données. Ces coûts restaient très important si bien que peu d'applications étaient déployées. En effet, les coûts étaient reportés sur les utilisateurs des services et seules les applications critiques ou à forte valeur ajoutée pouvaient être déployées sur ce modèle.

A l'instar du co-voiturage, le principe du cloud computing consiste à ne louer aux éditeurs qu'une partie du serveur. Le serveur étant mutualisé, les coûts de fonctionnement en sont très réduits. Ces fractions de serveurs sont appelés des *serveurs virtuels*. À la charge des fournisseurs de serveurs virtuels de maintenir l'infrastructure matérielle, et aux éditeurs logiciels de mettre en place un modèle économique pour leurs applications. Pour que ce modèle économique soit viable il est nécessaire que le coût de location des serveurs virtuels soit le plus faible possible.

Le coût d'un serveur virtuel peut se décomposer en deux parties. La première est liée à l'investissement (construction de la salle, achat du serveur etc.), la seconde est liée au fonctionnement. Le coût de fonctionnement d'un serveur est composé lui-même de deux parties. La première est liée à la maintenance du serveur (salaire des administrateurs, maintenance des logiciels de base etc.) et la deuxième liée à l'énergie électrique consommée par le serveur et par les équipements nécessaires à son fonctionnement (Climatisation, switch réseaux etc.). Cette part du coût de fonctionnement n'a cessé de croître depuis plusieurs années, d'une part parce que l'énergie électrique est toujours de plus en plus chère, et d'autre part car les serveurs n'ont cessé de se densifier en équipement.

Dans un centre de données, maîtriser les coûts de fonctionnement est une condition nécessaire pour proposer des serveurs virtuels à bas coût. L'objectif de cette thèse est de proposer une solution dédiée permettant la maîtrise énergétique des centres de données.

1.2 Une consommation électrique toujours grandissante

Les centres de données sont généralement sur-dimensionnés pour assurer la qualité de service des applications hébergées. Une étude réalisée par Google sur un de leurs centres de données, composé de 5 000 serveurs, décrit que le taux moyen d'utilisation CPU de la plupart des serveurs varie entre 10% et 50% [BH07]. D'un point de vue énergétique, cette sous utilisation entraîne des pertes électrique très importantes tant au niveau du serveur que sur l'ensemble du centre de données. En effet, la consommation électrique d'un serveur n'est pas proportionnelle à son activité : dans l'état *idle* sa consommation électrique représente 50% à 80% de sa consommation en pleine charge [DHKC09]. Non seulement les serveurs *idle* consomment de l'électricité sans valeur ajoutée, mais ils ont un impact important sur la consommation électrique globale du centre de données.

Issue d'une étude réalisée par IBM [ME09], la figure 1.1 décrit la consommation électrique de l'ensemble des éléments d'un centre de donnée classique. Pour 100 unités d'énergie, 55 sont utilisées pour le bon fonctionnement des serveurs (climatisation, générateur...). Les 45 restantes sont consommées par les serveurs. 70% de cette consommation des serveurs est utilisée par des équipements autres que le CPU (périphériques de stockage, ventilateur, conversion électrique ...). La consommation du CPU est donc réduit à 13.5 des 100 unités d'énergie initiales. Finalement, le serveur étant la plupart du temps en mode *idle*, seuls 3% de l'électricité consommée par un centre de données est utilisée pour exécuter les applications. 97% de l'électricité consommée par un centre de données est utilisée pour refroidir les serveurs (du ventilateur interne jusqu'au système de refroidissement de la salle), pour améliorer/garantir la chaîne électrique (onduleurs, redresseur...), maintenir le serveur en fonctionnement (par exemple dans l'état *idle*) ou utilisés pour les équipements annexes (des consoles d'administration aux néons éclairant la salle). De nos jours, le coût d'exploitation d'un serveur approche voir dépasse son coût d'achat [Bar05].

FIGURE 1.1 – Répartition de la consommation électrique d'un centre de données classique (IBM, 2009)

En vue d'améliorer le taux d'utilisation des serveurs, un grand nombre de centres de données ont déployé des systèmes de virtualisation, permettant d'offrir des serveurs virtuels. Ces systèmes permettent de consolider un ensemble de serveurs virtuels sur un

même serveur physique. En augmentant le taux d'utilisation des serveurs physiques, le ratio de capacité de traitement par watt consommé s'accroît. Si la virtualisation a effectivement permis d'augmenter ce ratio, elle n'a en revanche pas diminué le nombre de serveurs physiques dans un centre [GL10]. En effet, la virtualisation apporte une flexibilité dans l'administration des centres qui, comparée au déploiement de serveurs physiques, permet de déployer très facilement de nouvelles machines virtuelles. En conséquence, le nombre de machines virtuelles a fortement augmenté les cinq dernières année, ce qui a à son tour augmenté la densité des centres (via, par exemple, l'utilisation de serveurs lame) et en a finalement complexifié l'administration.

Or, augmenter la densité des centres de données impacte leur consommation électrique. Les centres de données actuels sont classiquement dimensionnés à 10-20 kW par armoire et jusqu'à 10-20 MW par centre. Pour une même surface au sol, les nouvelles infrastructures de type serveur lame entraînent une consommation de 200 MW par centre [Kat09]. Pour atteindre ces capacités, certains centres doivent modifier leurs systèmes d'approvisionnement en électricité (comme celui des impôts à Noisiel), et certaines entreprises (comme Microsoft, Google, ou Yahoo) doivent déployer les leurs près d'une source de production électrique abondante et peu coûteuse [Gra09].

Dans un système idéal d'*Energy-proportional computing*, l'énergie consommée par une application est nulle lorsqu'elle n'effectue aucun calcul et augmente linéairement avec la charge applicative. Dans le carde des centres de données, l'énergie consommée n'est pas liée uniquement au(x) serveur(s) exécutant l'application mais aussi aux équipements annexes à ces serveurs (climatisation, chaîne électrique...). Cette consommation est présentée dans la figure 1.2. Cette dernière présente la consommation électrique (en ordonnée) totale (des serveurs et de la climatisation) d'un centre de données homogène en fonction (en abscisse) de son taux d'utilisation et de divers algorithmes de placement de tâches entrantes, appelées aussi *jobs*. Cette courbe a été réalisée par l'équipe d'Impact Lab de l'Arizona State University [MBV+09], par simulation et met en évidence le rôle primordial des algorithmes de placement des *jobs* dans les centres.

La première courbe de ce graphique (frange 1), représente la consommation électrique optimale : la consommation électrique du centre est proportionnelle à la consommation des ressources par les *jobs*. Nous remarquons qu'à charge nulle le centre de données ne consommerait évidemment pas d'électricité tandis qu'à charge maximale, ce dernier consommerait une valeur maximale pour cette courbe.

A l'inverse, la dernière courbe représente la consommation au pire cas d'un centre de données. Ce pire cas correspond, lors de la montée en charge du centre de données, à un algorithme de placement de *jobs* choisissant les serveurs les plus consommateurs d'électricité et sollicitant le plus le CRAC (*Computer Room Air Conditioner*).

La frange numéro 2 de la figure 1.2 présente le gain obtenu en éteignant les serveurs non utilisés. Nous remarquons que ce gain est d'autant plus important que le centre de données est peu utilisé.

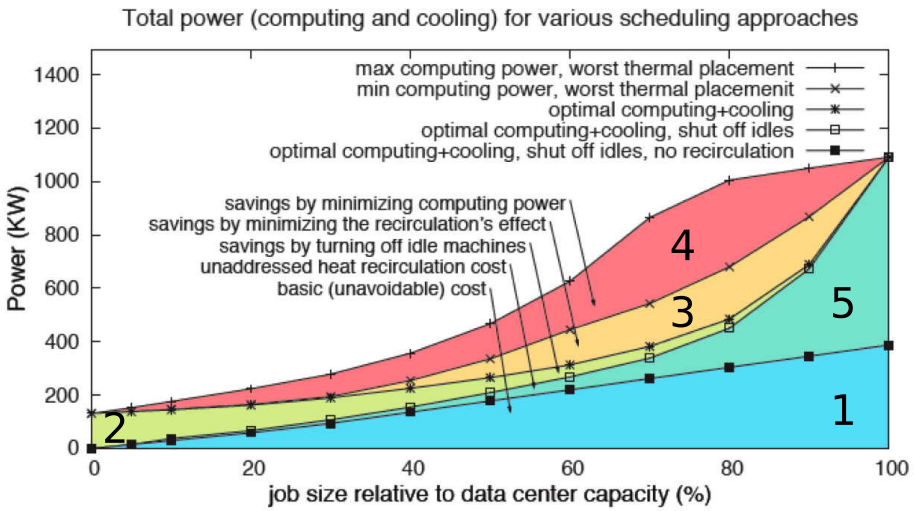


FIGURE 1.2 – Consommation électrique totale d'un centre de données

La frange numéro 3 représente le gain électrique avec l'utilisation d'un algorithme de répartition de charge thermique. Le principe de l'algorithme est de choisir en priorité, lors de la montée en charge, les serveurs sollicitant peu le CRAC.

La frange numéro 4 représente le gain énergétique pour un algorithme de placement prenant en compte la consommation électrique des serveurs, en affectant les *jobs* aux serveurs les moins énergivores.

La dernière frange 5 représente le gain potentiel en optimisant le placement des racks et serveurs via un réaménagement physique du centre de données.

Notre objectif est de proposer un système EPC (*Energy Proportional Computing*) complet, et plus spécifiquement un service de placement de serveur virtuel orienté énergie, s'appuyant sur les fonctionnalités de la virtualisation, prenant en compte l'énergie consommée par les serveurs (adressant la frange 4) et la dissipation thermique des serveurs (frange 3).

1.3 Une solution flexible, efficace et agnostique pour la maîtrise énergétique des centres de données

De nombreux travaux se sont appuyés sur la virtualisation pour développer des systèmes de consolidation dynamique des charges [MGW09], mais peu cependant ont intégré la dimension énergétique. Le système de consolidation développé dans le cadre de la thèse de Fabien Hermenier (BTRPLACE) a comme objectif de minimiser le nombre de serveurs utilisés, les serveurs inutilisés pouvant être éteints. Le développement de Snooze

quant à lui [FRM12] s'attache à passer à l'échelle le système de gestion en le distribuant dans le centre, répartissant ainsi la charge de calcul. L'initiative la plus aboutie est celle du Barcelona Supercomputing Center (BSC) et du Technical University of Catalonia (UPC) qui propose dans Emotive Cloud [AT09] un environnement permettant le développement d'ordonnanceurs prenant en compte la consommation électrique dans le processus de consolidation.

Dans le cadre de cette thèse, nous avons étendu le système Entropy pour :

- Le rendre modulaire et intégrer à la demande au système de placement des vues énergétiques, thermiques, électriques ou autres.
- Le rendre plus efficace en développant des heuristiques de placement au sein du solveur associés aux modèles de consommation des serveurs.
- Permettre de passer à l'échelle, en permettant d'adapter son comportement pour des problèmes de grande taille.
- Le rendre évolutif en lui permettant d'intégrer des modèles de serveurs à consommation autre que linéaire.

1.4 Diffusion scientifique

Ce travail a fait l'objet de plusieurs publications internationales et une nationale. Dans [IC-2] nous détaillons le système OptiPlace, un système étendant Entropy, plus efficace, plus flexible et permettant d'intégrer plusieurs modèles de consommation des serveur. Dans [IC-1, NC-1] nous présentons StressCloud, un logiciel dédié à l'injection de charge dans une infrastructure matérielle distribuée et virtualisée.

INTERNATIONAL CONFERENCE

[IC-2] Guillaume Le Louët and Jean-Marc Menaud. OptiPlace : designing cloud management with flexible power models through constraint programing. In *6th IEEE/ACM International Conference on Utility and Cloud Computing (UCC 2013)*, December 9-12, Dresden, Germany, 2013.

[IC-1] Guillaume Le Louët and Jean-Marc Menaud. StressCloud : An infrastructure stresser for Virtual Machine Managers. In *the first Energy Efficiency in Large Scale Distributed Systems conference*, Vienna, Austria, 22-24 April, 2013.

NATIONAL CONFERENCE

[NC-1] Guillaume Le Louët, Jean-Marc Menaud. StressCloud, un outil d'évaluation pour les gestionnaires d'infrastructure virtualisée. In proc. of *9ième Conférence Francophone sur les Systèmes d'Exploitation (CFSE09)*, France, Janvier 2013

Contexte

2

Gestion énergétique de l'informatique dans les nuages

Sommaire

2.1 L'informatique dans les nuages

Depuis la fin du siècle dernier, l'utilisation de l'outil informatique a suivi une augmentation fulgurante, tant par le grand public que par les entreprises. Les utilisations de cet

outil se sont multipliées, au point de le rendre indispensable dans la société actuelle. La messagerie en ligne, les réseaux sociaux, la modélisation météorologique, la simulation des réactions nucléaires, l'analyse des tendances d'un marché, la gestion économique d'une entreprise, l'achat en ligne, la publication de contenu multimédia etc. sont des utilisations dorénavant quotidiennes de l'outil informatique.

En particulier, l'utilisation de services à distancce s'est fortement développée, rendant l'accès à internet nécessaire pour beaucoup d'utilisations de l'outil informatique. Chacun de ces services à distance nécessite l'administration et la maintenance de leur puissance de calcul respective.

Cette puissance de calcul est présente physiquement sous la forme de serveurs informatiques. Ceux-ci exécutent, dans un environnement logiciel appelé OS, les programmes nécessaires pour fournir les services demandés. Le besoins en services informatiques étant en augmentation, les entreprises prestataires de ces services ont alors accru leur capacité de traitement. Cet accroissement a été réalisé via l'augmentation de la taille de leur parc de serveurs, mais celle-ci fit apparaître des problèmes liés au changement d'échelle. Par exemple, les problématiques liées au stockage des données, à la sécurité du matériel et à la résilience des services nécessitent des soins particuliers.

Ces entreprises sont amenées à investir massivement dans des salles dédiées à l'exécution des tâches informatiques. Ces salles sont communément appelées *centres de traitement des données*, et peuvent comporter des milliers de serveurs. Elles sont aménagées de manière à installer et administrer facilement non seulement les serveurs, mais aussi le matériel utilitaire du centre (switch, climatisation, etc.). Cependant, l'accroissement du nombre de serveurs dans ces salles soulève à son tour de nouvelles problématiques, telles la dissipation de la chaleur ou la question du taux d'humidité dans l'air.

Du fait de la diversité des contextes de l'informatique industrielle, l'installation et l'administration de tels centres sont donc des tâches délicates. En particulier, chaque centre a ses propres spécificités de fonctionnement. Par exemple, certains centres sont dédiés à l'hébergement de services spécifiques (HPC, base de données etc.) et donc optimisés dans ce sens. D'autres centres au contraires sont conçus pour l'hébergement de services variés, et aux performances variables. De par cette variabilité des services, les activités d'un centre à l'autre sont différentes. De plus, dans un centre les activités des serveurs sont variables, que ce soit parmi différents serveurs du centre ou au cours du temps pour un serveur.

Cette activité variable des serveurs est appelée *charge de travail*. Elle correspond à l'utilisation des ressources d'un serveur par rapport à ses capacités en ressources. Pour permettre une montée en charge des applications hébergées, l'administrateur veille à conserver une charge moyenne du centre faible. Même dans les centres de type HPC, où les services utilisent les serveurs au maximum de leurs capacités, des serveurs sont en attente de tâche à exécuter, donc sans activité. Les centres de données sont donc sur-dimensionnés en terme de capacité de calcul et d'infrastructure technique (dédié au

refroidissement, au réseau, au stockage).

Par soucis de rentabilité, des entreprises proposent de louer à des entreprises externes une partie des machines de leur centre (par exemple Amazon). Cette externalisation permet la mutualisation tant de la salle et du matériel que de la maintenance du centre, et donc du coût du centre. Suite à la croissance des besoins de tels centres, leur location est devenue une activité à part entière. Des entreprises se sont ainsi spécialisées dans l'hébergement de services informatiques. Pour l'entreprise cliente, délocaliser ses services réduit leur coût de maintenance et le besoin en personnel qualifié. Elle permet aussi d'améliorer la qualité des services, par exemple grâce à des infrastructures réseau ou de stockage dédiées, ou grâce à une expertise poussée.

Le cloud computing ajoute à cette notion de délocalisation des services une notion d'élasticité des ressources et de facturation à l'utilisation. Ces notions sont possibles grâce à la technique de virtualisation des ressources [AFG+10].

2.1.1 La virtualisation des systèmes informatiques

La technique de virtualisation des systèmes informatiques consiste à encapsuler l'environnement d'exécution d'un service dans un système virtuel(VM. Cette VM est elle-même exécutée sur un serveur grâce à un environnement logiciel spécifique appelé *hyperviseur*. Il est ainsi possible d'exécuter sur un même serveur plusieurs VM sans que celles-ci aient "conscience" les unes des autres. Cette technique permet d'améliorer le partage des ressources dans les centres.

En effet, dans les centres non virtualisés chaque serveur participe à l'exécution d'un seul service. Chaque serveur nécessite donc l'installation des programmes spécifiques à ce service. L'administrateur du centre doit réagir quand l'exécution d'un service sur un serveur lui demande trop de ressources de calcul. Il peut alors soit re-localiser les programmes sur un serveur plus puissant, soit répartir la charge des services sur un nombre plus important de serveurs. Dans les deux cas il doit disposer de nouveaux serveurs, les démarrer et leur installer les couches logicielles nécessaires. Il doit finalement paramétrer le centre pour prendre en compte cet accroissement des capacités de calcul. Cet accroissement des ressources mises à la disposition d'un service est très long à réaliser et donc peu pratique dans un centre de grande taille.

Lorsqu'au contraire le besoin de calcul d'un service diminue, les serveurs exécutant ce service réduisent leurs activités. Certains serveurs peuvent alors être éteints pour réduire la consommation du centre ou être utilisés pour l'exécution d'autres services. Cependant, au vu de la difficulté à configurer les serveurs et du temps nécessaire pour les redémarrer, il n'est pas toujours possible d'éteindre ou de réaffecter des serveurs. C'est pourquoi, dans les centres non virtualisés, la reconfiguration logicielle d'un serveur est une tâche exceptionnelle.

Des technologies de cohabitation des services permettent d'exécuter sur un même serveur différentes configurations d'un même programme. Ce serveur peut ainsi héber-

ger différentes configurations d'un même service, mais ces technologies apportent leurs propres inconvénients. Ainsi en cas de problème logiciel ou d'intrusion sur le serveur, ce sont les services de tous les clients qui sont menacés. De plus, la présence de plusieurs services sur un même serveur peut nécessiter le maintien de plusieurs couches logicielles. Cette cohabitation des services peut donc entraîner une complexité d'administration importante, pour des risques non négligeables.

Isolation et configuration

La virtualisation des services sépare les différents environnements logiciels de manière transparente. Ainsi, une VM atteinte par un virus ne contaminera pas les VM co-hébergées. La virtualisation apporte ainsi une solution efficace à la mutualisation des ressources informatiques pour l'hébergement de plusieurs services [FDF03]. Cependant, la présence d'un hyperviseur nécessite parfois une adaptation du système exécuté.

Idéalement, les services exécutés sur un système virtualisé doivent être les même que sur un système non virtualisé. Ceci permet d'assurer les mêmes fonctionnalités sans nécessiter de développement spécifique. Pour cela, l'hyperviseur simule un environnement matériel connu de la VM. Ceci est possible par la virtualisation dite *matérielle*. Dans cette approche, l'exécution et le contrôle des ressources d'une VM sont proposés par des circuits spécifiques du serveur, sur le CPU par exemple [WRF+10]. La virtualisation est actuellement intégrée aux CPU, permettant à l'hyperviseur de contrôler les droits d'exécution de code des VM. Cette technique permet l'exécution de logiciels sans connaissance de la virtualisation. Elle peut cependant être améliorée par l'ajout de code spécifique dans les VM, par exemple des drivers virtualisés ou le remplacement dynamique de parties du code système. Ces modifications de la VM peuvent être effectuées si l'OS de la VM est conscient de la virtualisation, donc configuré pour celle-ci. Les programmes métiers de la VM restent cependant les mêmes que sur un serveur classique.

Dans certains l'OS est connu mais il est impossible d'installer un code dans la VM, par contrainte de sécurité par exemple. L'hyperviseur peut néanmoins insérer dynamiquement un code dans la mémoire de la VM sans que les changements soient visibles de l'intérieur de celle-ci. Ce *tissage* de code se limitant aux VM pour lesquelles l'OS est connu, il ne peut être effectué sur des VM génériques. Il permet cependant d'améliorer les performances du centre sans nécessiter de configuration des VM lorsque les conditions s'y prêtent.

Sous réserve de quelques configurations de l'environnement logiciel, la virtualisation permet donc de mutualiser les ressources physiques pour l'exécution de services. Cette technique propose de plus de nouveaux concepts pour la manipulation des services d'un centre virtualisé.

Ressources matérielles et ressources virtuelles

La virtualisation des services permet de dissocier les serveurs physiques des services hébergés. L'administrateur garde le contrôle des VM exécutées sur chaque serveur, tandis que les clients du centre ont le contrôle des programmes exécutés dans leurs VM. Pour cela, un hyperviseur propose à une VM des ressources virtuelles nécessaires à son fonctionnement. Ces ressources peuvent être un ou plusieurs cœurs CPU ou GPGPU, un accès à la RAM, à des périphériques de stockage ou à des périphériques réseaux.

L'accès aux périphériques réseau et de stockage est un point crucial dans l'hébergement mutualisé. En effet, il est nécessaire que les programmes d'un client n'aient accès qu'au réseau de ce client, et qu'un service ne puisse modifier les données propres à un autre service. La virtualisation permet à l'hyperviseur de s'assurer de ces règles. D'une part, elle propose la création pour chaque client d'un réseau virtuel ne mettant en relation que ses VM. D'autre part, elle limite l'accès de chaque VM à un disque privé de stockage virtuel.

De la même manière que les accès au réseau et aux périphériques de stockage sont limités dans une VM, les accès à la RAM et au CPU sont restreints aux données propres à cette VM. De plus, l'administrateur d'un centre peut partager finement les ressources de calcul des serveurs parmi les VM. Il peut par exemple en attriber des priorités d'accès aux ressources, ou assurer un partage équitable de ces ressources. Il existe ainsi dans Xen et VMWare des mécaniques de réservation des ressources. Ceux-ci permettent d'attribuer une quantité minimale de CPU à une VM, quelque soit l'activité des autres VM exécutées sur le serveur. La virtualisation permet donc l'isolation des services, tant en terme de données qu'en terme de performances.

En plus de limiter les accès aux ressources, la virtualisation permet d'observer l'utilisation effective des ressources par une VM. Cette observation permet par exemple à l'administrateur de détecter les VM à comportement suspect, telle un serveur web qui monopolise le CPU. Elle permet aussi d'améliorer le comportement de VM grâce au partage des ressources virtuelles. Ainsi VMWare permet d'activer une option de recherche et de partage des pages mémoire identique parmi les VM. Lorsque plusieurs VM utilisant le même OS sont hébergées sur un serveur, il est probable que celles-ci utilisent les mêmes librairies logicielles. Cette technique permet ainsi de réduire globalement l'empreinte mémoire des VM hébergées sur un serveur.

Cependant, la virtualisation des ressources matérielles implique une surcharge CPU pouvant aller jusqu'à 30% [HLAP06] de la capacité du serveur. Comme les centres de type HPC nécessitent une capacité de calcul maximale, ils ne peuvent généralement pas se permettre une telle perte de performances. Bien que des techniques permettent de réduire cette surcharge, leur utilisation nécessite des paramétrages des VM qui ne sont pas toujours possibles. La virtualisation est donc réservée aux centres pour lesquels la dissociation des services et des serveurs est plus importante que les performances de calcul, par exemple ceux dédiés à l'hébergement de services génériques. En effet, la

virtualisation des services permet des opérations spécifiques dans les centres virtualisés.

Contrôle des machines virtuelles

L'administrateur d'un centre virtualisé dispose d'actions spécifiques à la manipulation des VM, du fait que celles-ci soient supervisées par un autre programme.

Il peut ainsi *suspendre* à tout moment l'exécution d'une VM, mémorisant l'activité de celle-ci au niveau des ressources virtualisées, et la redémarrer ultérieurement. Les systèmes Linux permettent déjà de suspendre l'exécution d'un processus et la reprise ultérieure. Cependant, cette action n'est pas permise quand le processus accède à certaines ressources, telles le stockage ou le réseau. Contrairement à ce mécanisme, toutes les ressources d'une VM sont virtualisées et donc manipulables par l'hyperviseur. La suspension d'une VM est donc possible quelque soit l'activité de celle-ci.

De plus, l'état d'exécution d'une VM, qu'elle soit en activité ou suspendue, peut être enregistré dans un fichier, créant un *snapshot* de celle-ci. Cet enregistrement d'état peut être ensuite utilisé, soit pour restaurer la VM dans son état précédent, soit pour créer une nouvelle VM à partir de cet état. Un snapshot comporte une sauvegarde de la mémoire et de l'activité CPU de la VM. Il mémorise aussi les modifications apportées aux périphériques de stockage de la VM depuis cet enregistrement. Il permet ainsi de retrouver non seulement l'activité de la VM mais aussi ses données dans l'état enregistré. Lors de la création d'une nouvelle VM à partir d'un snapshot, la conservation des données de la VM s'effectue grâce à la copie du disque virtuel.

Étant donné que le stockage est virtualisée, il est possible de partager un même disque virtuel entre plusieurs VM sans modification de celui-ci. En effet, la virtualisation permet d'attribuer différents modes d'accès à un périphérique de stockage, par exemple lecture seule ou lecture et écriture. Dans ce cas, le mode *copy on write* (pour copier quand modifié) est utilisé. Il stipule que les modifications apportées par une VM sont enregistrées dans un emplacement propres à cette VM. Cet emplacement étant temporaire, ces modifications sont perdues à l'extinction de la VM. Cette technique permet d'économiser en capacité de stockage. En effet, un seul disque virtuel est utilisé pour exécuter un grand nombre de VM simultanément. De plus, l'espace disque propre à chaque VM étant géré dynamiquement, cette approche permet d'optimiser l'espace de stockage. Cette technique permet de plus de créer très facilement de nouvelles VM, puisqu'il suffit de spécifier un modèle d'image disque à utiliser comme disque virtuel, en mode copy on write.

L'opération qui consiste à enregistrer puis copier directement l'état d'une VM peut aussi être effectuée en une seule fois, créant un *clone* de celle-ci. L'opération de clonage permet d'augmenter facilement, dans les architectures le permettant, le nombre de VM participants à un service. En effet, au lieu de devoir installer puis démarrer une couche logicielle sur un serveur (virtualisé ou pas), cloner une VM produit une nouvelle VM déjà active.

Ce clonage peut être réalisé en une seule fois pour créer une nouvelle VM indépendante où être maintenu dans le temps. Ainsi dans les centres VMWare, la fonctionnalité de *High Availability*, pour Haute Disponibilité, consiste à synchroniser en temps réel l'état d'exécution d'un clone avec sa VM originelle. Si la VM tombe en panne, par exemple par l'extinction de son hyperviseur ou un problème de réseau localisé, alors la VM clone prend le relais. Lorsque cela arrive toutes les connections de la VM originelle sont redirigées vers la VM clone. Cependant, si cette technique permet d'assurer une continuité des services malgré des problèmes d'infrastructure, elle nécessite un réseau à haute performance. Elle requiert en effet de propager en permanence les modifications de la RAM et du CPU d'une VM vers l'autre.

À ces opérations propres à la création et la manipulation des environnements virtualisés s'ajoute la capacité pour l'administrateur de changer l'hôte d'une VM.

Migration des machines virtuelles

Un service virtualisé est exécuté dans une VM, elle-même hébergée sur un serveur. Du fait de cette séparation entre ressources physiques et virtuelles, les VM peuvent être déplacées, avec ou sans interruption de service, d'un serveur à un autre. Cette technique de déplacement des services s'appelle la *migration*, à froid ou à chaud, des VM.

La *migration à froid* d'une VM consiste à suspendre la VM, puis à copier ses plages mémoires et son état d'exécution depuis le serveur hôte vers le serveur destination, et enfin reprendre l'exécution de la VM. Cette opération permet de déplacer une VM d'un serveur à l'autre mais nécessite l'interruption des services, qui impossible pour des services critiques.

La migration à chaud d'une VM est proche de la migration à froid, cependant la copie des données de la VM se fait sans interruption de celle-ci. Pour cela, une partie de la mémoire est copiée vers le serveur hôte durant l'exécution de la VM, puis cette VM est interrompue durant un laps de temps très court, durant lequel l'état exact d'exécution de cette VM et les connexions réseau sont transférés vers le nouveau serveur hôte. Une fois cet état copié, l'exécution de la VM est alors reprise par le nouveau serveur hôte. En particulier, durant une migration à chaud, l'intervalle de suspension de la VM est suffisamment faible pour ne pas rompre les connections TCP/IP. Ainsi, si l'infrastructure le permet, une migration à chaud n'est pas perceptible de l'intérieur de la VM.

Cette migration à chaud permet de démarrer les VM sur des serveurs très rapides, puis de les migrer sur des serveurs plus économes afin d'exécuter au plus tôt les demandes de création de VM personnalisées par les clients. Elle permet aussi d'accroître les ressources à disposition d'une VM, tel le nombre de cœurs CPU disponible, en la migrant vers un hôte à capacité plus importante, ou bien au contraire de réduire le nombre de serveurs démarrés, en déplaçant les VM depuis les serveurs sous-utilisés sur un un seul serveur.

Cette technique de virtualisation permet ainsi de déployer plus facilement des ar-

chitectures orientées services multi-tiers, tout en réduisant les empreintes mémoire et disque, ainsi que le nombre de serveurs nécessaires. Elle permet aussi d'assurer un niveau de qualité pour les services hébergés, par la duplication des données et de l'exécution ou l'utilisation de snapshots automatiques. Des infrastructure à haute disponibilité pour l'hébergement de services virtualisés peuvent ensuite être utilisées pour offrir des services d'hébergement à qualité contractuelle.

2.1.2 De la virtualisation aux XaaS

La virtualisation est une technique permettant de dissocier un service de l'infrastructure physique dans laquelle il est exécuté. En s'appuyant sur cette notion, les industriels ont proposé de nouveaux modèles d'hébergement des services, reposant sur des notions tant techniques que commerciales. Ces modèles d'utilisation des infrastructures sont utilisés dans le cloud computing, et réparties en trois catégories distinctes : IaaS, PaaS et SaaS. Ces modèles sont décris ci-après.

Location d'infrastructure

Le niveau le plus bas est le IaaS, c'est à dire l'hébergement de machines virtuelles sur les serveurs du centre. À ce niveau, le centre fournit à ses clients une infrastructure virtuelle permettant l'exécution des VM à la demande. Cette infrastructure comporte bien sûr des serveurs pour héberger les VM, mais aussi le réseau de communication entre les VM, parfois avec internet, et un système de stockage des données.

Un centre fournissant un service de IaaS peut soit proposer des VM pré-configurées dans lesquelles les clients installent leurs programmes, soit accepter l'exécution de VM créées par les clients. le client d'un tel centre est responsable de la configuration de ses VM et de l'architecture virtuelle du réseau. L'administrateur du centre est quant à lui responsable de la bonne exécution des VM dans l'infrastructure proposée.

Typiquement, le service d'Amazon EC2 permet aux clients de démarrer de nouvelles VM, soit créées entièrement, soit à partir de modèles, et d'y exécuter les programmes de leur souhait. Plusieurs types d'hébergements sont proposés, permettant d'optimiser les ressources virtuelles en fonction de l'usage qui est prévu de chaque VM. Le client sélectionne ses VM, la classe de ressources virtuelles qui lui seront accordées, puis est facturé pour chaque heure d'exécution de ses VM. Lorsqu'un client a besoin de plus de capacités de calcul, il peut réserver et démarrer de nouvelles VM, ou bien demander à ce que cette modification soit automatique.

Ce sont ces centres que nous considérons dans cette thèse, car ils sont le socle de base du cloud computing.

Location d'une plate-forme de services

Les centres de type PaaS se positionnent au-dessus des IaaS en terme d'abstraction des services. À ce niveau, un tel centre propose à ses clients un environnement de services travaillant conjointement. Les clients y hébergent leurs applications en s'appuyant sur cette plate-forme de services.

Une telle plate-forme propose des services basiques tels le stockage des fichiers, le support d'une base de données ou l'hébergement de sites web. Elle peut aussi proposer des services propres à une certaines activité, telle le développement logiciel. Ainsi, la plate-forme WINDOWS AZURE héberge des serveurs pour le jeu TITANFALL, déchargeant les consoles connectées d'une part de leurs calculs. Outre WINDOWS AZURE, la plate-forme d'applications web GOOGLE APP ENGINE permet aux clients d'installer leurs applications en tant que scripts. Elle met à disposition les services de GOOGLE, tels la gestion des mails ou l'accès à une base de données, réduisant le besoin de configuration des clients.

Au niveau d'un PaaS, la virtualisation des services n'est plus visible par les client qui n'ont accès qu'aux services mis à disposition par la plate-forme. L'élasticité des ressources offerte par un PaaS permet aux développeurs de ne pas se soucier de la capacité à passer à l'échelle. De même, la présence de services assurés par des entreprises externes réduit le temps de développement des développeurs. Ainsi, un PaaS est particulièrement adapté pour les start-up, qui ont besoin de proposer rapidement de nouveaux services à leurs clients.

Si le PaaS offre des facilités de déploiement d'applications, il est cependant plus restrictif que le IaaS. En effet, certains services offerts par un PaaS peuvent lui être exclusifs ou utiliser un protocole fermé. Les développeurs utilisant un tel service devront adapter une partie de leur code en cas de changement de plate-forme. De plus, les systèmes de stockage des données peuvent manquer de sécurité. Ainsi le service de mail fourni par GOOGLE analyse les mails transmis, posant un problème de confidentialité.

La tendance des entreprises est d'héberger leurs applications sur un PaaS dans un premier temps. Lorsque les services offerts sont bien définis, ces entreprises les isolent et migrent, au moins partiellement, vers un IaaS dans lequel elles peuvent contrôler leurs données et services plus finement. Les entreprises adaptent ainsi l'environnement qui permet d'exécuter les requêtes client.

Exécution des requêtes client

Le dernier niveau du XaaS est le SaaS, ou niveau des utilisateurs finaux. À ce niveau, les utilisateurs envoient des requêtes aux applications hébergées dans le centre. Les clients d'un tel modèle sont donc les utilisateurs des applications déployées dans le centre.

Le SaaS suppose l'installation d'une plate-forme sous-jacente pour proposer ses services. Ainsi, les éditeurs de solution on-line, tels MICROSOFT et sa suite office 365,

Yahoo et son service de mails ou encore le système de stockage en ligne et partage de Dropbox, peuvent proposer un service client performant, grâce à une plate-forme de services résiliente et à haute disponibilité.

Ces trois niveaux sont complémentaires et dissociables. Dans un centre virtualisé, le IaaS fournit au PaaS les ressources nécessaires pour supporter la couche logicielle, tandis que le PaaS est configuré par un administrateur pour offrir ses services aux clients sous la forme d'un SaaS. Un des points communs de cette centralisation est la mise en place de contrats de qualité de service avec les prestataires, objet de la section suivante.

Contrats de qualité de service

Une *bonne* exécution d'un service est nécessaire pour la satisfaction des clients finaux. Par exemple, en 2007 une augmentation du temps de réponse des requêtes du moteur de recherche GOOGLE de $500ms$ réduisait les revenus de 20% [KHS07]. Les clients d'un centre XaaS doivent donc avoir l'assurance que les services auxquels ils souscrivent sont correctement fournis.

Lors de l'utilisation d'un XaaS, le prestataire de service établit un contrat de service avec son client [KPR09]. Ce contrat contient en particulier des spécifications de qualité de service avec des clauses portant sur la présence de support client, des clauses financières, mais aussi des clauses de qualité d'exécution du service. Ainsi, en 2014 le service d'hébergement fourni par le IaaS GOOGLE Compute Engine spécifie des réduction du coût d'hébergement en cas d'activité insuffisante du centre (moins de 99.95% de disponibilité mensuelle).

Cette notion de qualité de service est très variable puisque chaque service proposé est différent. Ainsi, à chaque service correspond ses propres métriques de qualité, même si le taux de disponibilité, c'est-à-dire le pourcentage de temps où le centre peut répondre aux requêtes, est une métrique communément utilisées. Schématiquement, le contrat d'un IaaS spécifie des ressources allouées aux VM, tels les cœurs CPU, la mémoire ou la capacité de transfert disque, et des contraintes de placement des VM, telles l'éloignement de VM redondantes ou le placement dans un pays particulier. Au niveau PaaS et SaaS les métriques de performances portent sur des nombres de requêtes, les volumes de données et les délais de réponses aux requêtes spécifiques aux services fournis. Cependant, les services hébergés dans les PaaS et les SaaS étant très variables, leurs contrats de services le sont aussi.

Afin d'assurer une certaine qualité de service à ses clients, un hébergeur de services internet doit disposer d'une capacité de calcul suffisante pour amortir les pics de charge du centre. Pour cela, il peut sur-provisionner le centre, ce qui induit une perte d'efficacité moyenne du centre puisque des ressources doivent être disponible en permanence. Il peut aussi changer de paradigme et adapter ses capacités de calcul à la demande, ce qu'on appelle l'élasticité du centre.

2.1.3 Cloud Computing

Le cloud computing est un paradigme d'exécution des services informatiques[BYV+09]. Il se base sur un système de XaaS, c'est-à-dire de déportation des calculs informatiques du prestataire, en vue de répondre aux besoins de ses clients, tout en tirant partie de la virtualisation des tâches pour ajouter une notion d'élasticité et un modèle de facturation à la demande.

Principes

Le cloud computing consiste en l'utilisation de serveurs délocalisés pour exécuter des services, avec une quantité de travail nécessaire évaluée et facturée de manière dynamique par le gestionnaire du cloud. Il ne s'agit plus de payer pour la location d'un serveur physique permettant d'héberger des services, mais pour une certaine qualité d'hébergement et une infrastructure adaptable au besoin.

Ainsi, des critères tel la modularité de l'offre d'hébergement, la quantité de ressources disponibles dans le centre pour faire face à un accroissement d'activité, la capacité d'action dans le centre même, sont propres à chaque centre et justifient un nombre très important de fournisseurs de cloud computing. Une telle infrastructure repose, dans le cas du IaaS, sur la virtualisation des services, et donc l'utilisation de centres virtualisés.

Nous considérons dans cette thèse les problématiques d'administration logicielle de tels IaaS.

Complexité d'administration des IaaS

La gestion du placement des VM dans un centre virtualisé est un problème très complexe. Premièrement, quand le nombre de VM dans un centre est important, la vision globale du centre n'est plus facilement accessible. Des centres simples de quelques serveurs et dizaines de VM peuvent être administrés manuellement. Des centres de plus grande taille, tels ceux d'Amazon [Ham11], hébergeant des dizaines de milliers de serveurs ne peuvent plus l'être. À une telle échelle, l'administration manuelle n'est plus possible et l'administrateur du centre doit faire appel à des systèmes automatisés.

Ces systèmes de gestion doivent réagir rapidement, d'une part car les VM ajoutées au centre doivent être démarrées au plus tôt, et d'autre part car les VM sont sujettes à des évolutions fréquentes d'activité. Ainsi chaque site web, chaque serveur de flux, de base de données hébergé par une VM a des pics de fréquentation dans une même journée, dans une même semaine, dans un même mois. Si le placement des VM dans le centre n'est pas adapté à ces pics de fréquentation, il est nécessaire de reconfigurer très rapidement les VM du centre.

Cette nécessité de vivacité du système de gestion doit de plus faire avec un nombre parfois important de règles spécifiques au centre. Ainsi, des VM travaillant conjointe-

ment pour fournir un service devraient être placées sur le même serveur. Au contraire, une VM clone ne devrait pas être placée sur le même serveur que celle qu'elle clone pour des raisons de fiabilité. La compréhension de ces contraintes et leur prise en compte lors de la reconfiguration d'un centre nécessitent des travaux importants lors du développement des gestionnaires de IaaS.

Le fer de lance d'un centre de cloud computing se base sur trois critères : sa fiabilité, ses fonctionnalités et le prix d'hébergement des VM. Dans un marché en forte croissance avec la présence de grands acteurs tel que Amazon, le prix d'hébergement d'une VM est un des critères de vente les plus importants. Hors, le prix d'une VM se détermine en fonction de deux éléments : le coût fixe lié à l'investissement et le coût variable lié au fonctionnement de l'infrastructure. Si le prix des serveurs a baissé ces dernières années, le prix de l'énergie n'a cessé d'augmenter. Ainsi, il est estimé que de nos jours le prix d'un serveur est lissé sur trois ans, période sur laquelle ce prix est dépassé par le coût de fonctionnement du serveur. Cette thèse se propose d'optimiser le coût de fonctionnement. La section suivante fait un point sur la consommation électrique d'un centre de données.

2.2 Maîtrise énergétique

La maîtrise énergétique d'un centre concerne toutes les problématique liées à la consommation électrique du centre. Ces problématiques sont par exemple la réduction de l'empreinte énergétique globale, la limitation de la consommation des serveurs, ou encore l'évaluation des consommations des VM. Bien que ces problématiques soient récentes, la maîtrise énergétique des ordinateurs est un problème connu depuis longtemps.

2.2.1 Le calcul et l'énergie

La maîtrise de la consommation du matériel informatique est un problème qui date des premiers ordinateurs. C'est cette maîtrise, en particulier grâce à l'invention du transistor en 1947, qui a permis de manipuler des informations en quantité importante, à une fréquence intéressante [Lau83]. Elle a de plus permis l'utilisation de l'informatique dans des contextes non plus industriels ou de recherche mais aussi chez les particuliers, grâce à une consommation électrique suffisamment faible pour être acceptable dans un foyer.

Par la suite, les ingénieurs n'ont cessé d'inventer de nouvelles techniques pour réduire la consommation des serveurs.

Activité et consommation

Il y a deux sources de consommation électrique dans un ordinateur, la consommation de base et la consommation dynamique.

Premièrement, un circuit a besoin d'une tension minimale pour se maintenir dans un état stable, et cette tension induit des fuites énergétiques. Dans les processeurs cadencés par exemple, la présence d'une horloge envoyant une tension périodique implique une consommation de base pour uniquement assurer la réponse du CPU aux événements du serveur.

Deuxièmement, le traitement des données informatiques consiste à effectuer des opération sur des données et à en stocker le résultat. Que ce soit entre autres pour l'émission de tampons réseau sur un canal, pour le calcul d'une différence entre deux nombres entiers, pour l'envoi sur le bus de données pour la carte graphique, pour la vérification des données enregistrées dans la RAM, ou la distribution d'un fichier parmi plusieurs disques en RAID, la manipulation de données induit un accroissement de la consommation électrique du système. La transmission d'informations entre deux éléments d'un ordinateur ou entre deux ordinateurs peut être intégrée dans cette consommation dite *dynamique* du serveur.

Idéalement, à une consommation maximale donnée d'un serveur, la consommation réelle de celui-ci serait proportionnelle au travail qu'on lui demande. Ainsi, la consommation de base du serveur serait nulle tandis que la consommation dynamique serait linéaire avec chacune des activités possible du serveur. Comme mentionné, ce modèle idéal n'est pas atteint en pratique même si des techniques permettent de réduire la consommation de base du serveur, comme le clock gating des CPU [JMSN05].

Le remplacement des lampes par des transistors a permis de réduire énormément la consommation statique d'un circuit, en ne consommant de l'énergie qu'à chaque changement d'état. Depuis, les constructeurs n'ont eu cesse d'améliorer l'efficacité énergétique de leurs produits, afin d'améliorer en particulier leurs performances [HAP+05]. En effet, bien qu'il soit possible de créer des puces avec un nombre très important de transistors pour améliorer la capacité de traitement de celles-ci, les fondeurs sont limités (entre autres) par la capacité de dissipation du matériau, et donc par la fragilité du produit face à une hausse de température qu'une consommation importante génère. De même, si une augmentation du voltage d'un circuit permet d'augmenter la réactivité, et donc la fréquence de celui-ci, elle augmente aussi la quantité de chaleur générée par l'activité du circuit, et demande donc des systèmes de refroidissement plus importants, tels que les refroidissements à eau ou azote.

Miniaturisation des composants

Cette problématique de dissipation a été améliorée grâce à la réduction des composants. Ainsi, la taille des transistors des micro-processeurs est passée successivement de $10\mu m$

en 1971, à $1\mu m$ en 1989, puis sous la barre des $100nm$ dans le début des années 2000, pour arriver à $22nm$ dans les processeurs *Ivy Bridge* d'Intel en 2012. Cependant, les transistors ont actuellement une taille minimale pour fonctionner, aussi la miniaturisation des puces est arrivée à ses limites dans l'état actuel des connaissances. Bien que des technologies puissent améliorer encore la miniaturisation des circuits[Hir05], telles les transistors multi-portes[Col08], les constructeurs ont cherché un autre angle d'approche pour améliorer leurs systèmes.

Le pipelining

La solution pour augmenter la puissance des circuits à haute densité a été la parallélisation des traitements, c'est-à-dire l'exécution de plusieurs traitement en même temps. Cette parallélisation était déjà présente dans les CPU : le pipeline du processeur charge plusieurs instructions à exécuter, ainsi plusieurs instructions sont en cours d'exécution à un moment donné. Cependant, cette technique est limitée aux instruction d'un seul programme et atteint vite ses limites, en particulier elle augmente la consommation du CPU.

Unités de calcul dédiées

Une autre possibilité pour améliorer les capacités de calcul d'un ordinateur est de lui adjoindre des circuits dédiés à l'exécution de certains types de programmes, et donc plus efficaces pour certains traitements. Ainsi, depuis les années 1980 les ordinateurs ont des cartes graphiques(GPU) qui permettent de calculer l'affichage d'une scène graphique, réduisant la charge du CPU. En 2007 les langages CUDA et OpenCL permettent d'exécuter des programmes à taux important d'instructions parallèles sur un GPU compatible, sans que ces programmes ne produisent d'affichage. Le GPGPU est depuis une tendance forte de la modélisation informatique [YPZL05, MV08, WL08, SHUS10], et bénéficie des avancées en terme de parallélisation des unités de calcul.

Génération multi-cœurs

Un processeur "multi-cœurs" est un processeur qui contient plusieurs unités de calcul, les cœurs. Contrairement à l'addition d'unités dédiées à certains traitements, ces unités sont équivalentes en terme de capacité de traitement (bien que potentiellement hétérogènes [KFJ+03]), et capables de s'échanger l'exécution d'un ou plusieurs programmes.

En 2001 les premiers processeurs multi-cœur ont vu le jour, cette technique a été intégrée en 2005 dans les ordinateurs grand-public. Depuis, elle est présente dans la plupart des processeurs d'ordinateurs de bureau. L'avancée des technologies permet dorénavant d'utiliser de tels processeurs dans l'informatique mobile, où depuis 2011

les processeurs des smartphones peuvent être multi-cœurs, malgré l'accroissement de consommation inhérente à cette technologie.

En effet, si les calculs sont répartie sur plusieurs cœurs, la complexité induite par l'architecture multi-cœur réduit l'efficacité énergétique du processeur. Cette complexité est due à la synchronisation des données entre les différents cœur et le reste du système. L'adaptation dynamique de la consommation[HF05] permet de réduire ce défaut et dans certains cas de le compenser.

Ce parallélisme est nécessaire pour accroître la capacité de calcul des processeurs, la densité de transistors par unité de calcul étant difficilement améliorable. Le nombre d'unités de calcul par ordinateur, et donc de transistors par ordinateurs, augmente avec le temps, et la taille des transistors ne compensant pas cette augmentation, la consommation de ces ordinateurs augmente aussi. Cet accroissement de la consommation par ordinateur implique des systèmes de refroidissement plus efficaces, ce qui n'est pas toujours possible.

Le calcul distribué

Après la réduction de la taille des transistors pour avoir plus de transistor par circuit, la diversification des unités de calcul dans un ordinateur pour augmenter sa capacité de calcul, la solution pour accroître la puissance disponible à la résolution d'un problème est l'utilisation de grilles de calcul pour résoudre ce problème. Une telle grille consiste en un réseau de plusieurs ordinateurs, ou nœuds, travaillant conjointement à l'accomplissement d'une tâche. Ils exécutent donc des programmes ayant une connaissance de ce réseau, connaissance totale ou partielle dans le cas de nœuds esclaves.

Cette notion, bien que présenté depuis plus d'une décennie [CADAD$^+$97], demande de développer des programmes orientés calcul distribué, c'est-à-dire capables de scinder un problème en sous problèmes. La résolution des sous-problème, distribués parmi les nœuds du réseau, permet ainsi de résoudre le problème principal. Cette contrainte demande un changement de point de vue du développeur, mais est cependant déjà partiellement intégrée lorsqu'un programme veut bénéficier de systèmes multi-cœurs. En effet, les modifications d'un logiciel nécessaire à l'utilisation de plusieurs cœurs sont aussi nécessaires pour l'utilisation de plusieurs nœuds.

C'est cette approche distribuée qui est utilisée dans les modèles de cloud computing, où l'accroissement de la quantité de serveur permet d'augmenter dynamiquement la capacité de calcul disponible. Dans les grands centres de type IaaS, le coût d'installation des serveurs et le coût de la maintenance, tant logicielle que physique, par des techniciens et développeurs sont importants. De plus, la maîtrise de la consommation électrique est nécessaire pour maintenir un environnement viable pour l'activité du centre. Nous commençons par détailler les notions utilisées dans le cadre de ces problématiques.

2.2.2 Terminologie

Dans les centres de données, la consommation totale du centre sur un intervalle (par exemple un mois) est exprimé en $MW.h$. Cette valeur correspond à une différence de consommation d'énergie entre deux moments, donc à un effort électrique fourni, et permet d'exprimer l'intensité du travail réalisé durant cet intervalle. La consommation électrique des éléments du centre est toujours croissante, car le centre consomme et ne restitue pas d'électricité, bien qu'elle puisse être nulle quand le centre utilise des énergies vertes.

Afin de comparer deux relevés énergétiques, que ce soit pour les consommations d'éléments différents ou pour une différence de consommation dans le temps, il faut prendre en compte l'énergie consommée mais aussi l'intervalle de mesure de cette consommation. Aussi, nous utilisons la notion de puissance moyenne consommée par un objet, celle-ci étant le quotient entre la différence d'énergie consommée et la durée de l'intervalle. Ainsi, une puissance moyenne d'$1W$ correspond autant à une consommation d'$1J$ pendant $1s$, que de $3600J$(=$1W.h$) pendant $1H$.

La consommation instantanée d'un objet est relevée par des outils de mesure, cependant une précision très important n'est pas nécessaire. En effet, la consommation énergétique d'un CPU est liée à l'exécution des programmes, ce qui peut induire des différences notables de consommation entre deux intervalles de temps très courts, par exemple des pics locaux de consommations lors des tics d'horloge.

Si le système de mesure observe l'activité énergétique du serveur avec une précision trop importante, il observera de très grands écarts de consommation sur une très faible durée. De fait, une précision trop importante dans l'évaluation de la puissance consommée, de l'ordre de la fréquence d'horloge, réduit l'intérêt de cette observation.

Enfin, une fréquence d'observation trop importante peut augmenter la charge du système, et donc augmenter la consommation observée, quand le système observateur est aussi le système observé, comme c'est le cas pour les serveurs à sonde énergétique embarquée. Ceci peut être dû à des interférences électriques lors de l'observation dues à l'utilisation d'une sonde elle-même consommatrice d'énergie, ou à cause d'un coût de traitement des informations non négligeable.

Ainsi, et contrairement à l'intuition, une augmentation de la précision n'est pas toujours intéressante, la gestion des centres se satisfait d'observation réalisées avec une précision de l'ordre de quelques secondes. Nous utilisons généralement la notion d'énergie pour quantifier le coût d'une action ou de l'exécution d'une VM de type HPC, tandis que la notion de puissance nous permet d'évaluer des stratégies de gestion d'un centre soumis à un travail continu, par exemple des VM de type webserver.

2.2.3 Mesure énergétique dans un centre virtualisé

Le travail d'optimisation d'un centre nécessite des observations des consommations dans celui-ci. Ces mesures de consommation peuvent être réalisées à plusieurs niveaux.

La consommation totale du centre est relevée par le prestataire électrique, de manière régulière mais sur de longues périodes, par exemple une fois tous les six mois. Prenant en compte l'ensemble du centre, ils ne permettent pas de déterminer avec précision la consommation d'un serveur à un instant donné ni d'observer une hausse de consommation ou d'en comprendre la raison.

L'administrateur d'un centre doit donc déterminer les causes de consommation dans le centre et quantifier ces consommations, nous énumérons ici les différents éléments d'un centre qui participent à cette consommation.

Mesure au niveau du serveur

Dans un centre, la part la plus importante de la consommation est généralement attribuée au fonctionnement des serveurs pour exécuter leurs tâches. Il est donc opportun de mesurer cette consommation, pour déterminer par exemple les serveurs consommant le plus d'électricité.

Pour cela, les serveurs industriels intègrent généralement des sondes énergétiques interrogeables à distance. Ces sondes nécessitent cependant des pilotes adaptés, qui peuvent être compliqués à installer. Si des normes d'accès aux sondes des châssis se développent (telle IPMI), les implémentations de ces protocoles sont propres à chaque constructeur et nécessitent parfois des développements spécifiques. Dans les faits, l'observation des consommations des serveurs demande la configuration du pilote de chaque serveur, qui peut être délicate, voire même le développement de ce pilote. En plus des problèmes de pilotes, cette configuration peut être compliquée par la présence de multiples sondes dans un serveur, nécessitant à l'administrateur de déterminer laquelle est la plus intéressante.

Lorsque des serveurs ne mettent pas à disposition un accès à ces sondes pour l'administrateur ou que les accès présents ne sont pas satisfaisants, celui-ci peut installer des sondes externes. Celles ci sont placées entre l'alimentation du serveur et les prises électriques du centre. Dédiées à l'observation des consommations, elles sont généralement plus facilement accessibles et configurables que les systèmes intégrés aux serveurs. De plus, l'installation d'un ensemble de sondes du même modèle permet à l'administrateur d'homogénéiser, d'une part les accès à ces sondes, et d'autre part la sémantique des valeurs de consommation. Cette installation peut cependant nécessiter l'extinction des serveurs du centre, ce qui n'est pas toujours possible lorsqu'un serveur ne possède pas de double alimentation. De plus, l'utilisation d'un nombre important de sondes peut produire des interférences ou ne pas passer à l'échelle, par exemple en cas d'utilisation de sondes sur un réseau sans fil.

L'administrateur utilise ces sondes pour observer les consommations des serveurs, et en déduire des données statistiques. Ces données peuvent être utilisées pour aider au placement des VM sur les serveurs, par exemple en plaçant les VM en priorité sur les serveurs les plus efficaces. Pour cela, l'administrateur doit mettre en relation les activités des serveurs avec leur consommation, en paramétrant des modèles de consommation grâce à ces données. L'utilisation de modèles plus complexes permet généralement de réduire l'erreur de modélisation, mais demande plus d'observations avant d'être réalisable. Enfin, la sémantique des relevés, par exemple la position des sondes, et la granularité de ces relevé de consommations influent grandement sur ces modélisations.

Armoires et groupes de serveurs

Outre les serveurs, les modèles les plus avancés d'armoires dans lesquels ceux-ci sont agencés proposent des relevés de consommation du matériel de l'armoire. Dans ce cas, l'administrateur a accès à la consommation du centre à un niveau intermédiaire entre la consommation de chaque serveur et celle du centre dans son ensemble. Cependant ces données ne permettent pas d'agir directement sur la consommation des serveurs.

L'observation des consommations des serveurs pose de plus problème lorsque ces serveurs partagent des ressources. ainsi, des modèles récents de serveurs de type *serveur en lame* sont incorporés dans un châssis qui regroupe plusieurs de ces serveurs. Un tel châssis mutualise des fonctionnalités parmi les serveurs, telles l'alimentation électrique, la gestion du réseau, le stockage des données, réduisant ainsi non seulement la taille des serveurs, mais aussi les besoins d'administration. Ces châssis permettent d'accéder à des sondes énergétiques, mais si les consommations de chaque serveur lame sont accessible, il faut aussi répartir la consommation du châssis sur ces serveurs, celle-ci étant mutualisée parmi ces serveurs.

Les serveurs ne sont pas les seuls éléments consommateurs d'électricité dans le centre et la facture électrique totale prend en compte les éléments non dédiés au calcul du center.

Matériel non calculateur

Outre les serveurs, un centre contient du matériel utilitaire consommant de l'électricité : d'une part pour assurer le bon fonctionnement des serveurs du centre, tels les onduleurs, routeurs, et le système de stockage, et d'autre par pour assurer le bon environnement de travail, tels les systèmes d'observation ou de climatisation. Certains de ces éléments permettent d'accéder à leurs valeurs de consommation.

Les onduleurs d'un centre sont nécessaires pour lui assurer une sécurité en cas de micro coupures et permettent d'observer la puissance qu'ils fournissent au centre. Cependant, s'ils permettent d'avoir une vision globale de la consommation dans le centre,

il n'est pas possible de déterminer quels sont les éléments du centre responsables de cette consommation.

La consommation des routeurs et périphériques de stockage (NAS) est considérée comme très faible par rapport à celle des serveurs, elle est donc souvent négligée. De plus, les consommations de ces éléments peuvent être très difficiles à modéliser, par les nombreux paramètres à prendre en compte. Par exemple, l'activité d'un routeur dépend des activités réseaux des serveurs du centre, mais aussi de la configuration des réseaux virtuels. De même, l'activité des périphériques de stockage dépend non seulement des activités des serveurs, mais aussi des politiques de sauvegarde, de cache ou de compression qui sont utilisées. La consommation de ces éléments étant très peu variable par rapport à la complexité de leurs modèles, elle est généralement considérée comme constante, et donc incorporée dans la consommation de base du centre. Nous ne prenons donc pas ces éléments en considération.

Le système de climatisation d'un centre peut être une simple ventilation d'air extérieur, purifié et acclimaté pour réduire l'usure des serveurs. Cependant, les variations de température importantes de l'air extérieur nécessite généralement la régulation de la température du centre. Si cette régulation est importante, ce système de climatisation peut compter pour une part non négligeable de la consommation du centre. Il est donc intéressant lors de la création et de l'administration d'un centre de grande taille de prendre en compte les problématiques de flux thermiques, qui peuvent être sources de sur-consommation du centre, comme présenté dans [PBB+01].

L'impact énergétique du matériel d'un centre est représenté par un indicateur d'efficacité appelé PUE, pour Power Usage Effectiveness [BM07]. Cet indicateur est calculé par le ratio entre la consommation totale du centre et la consommation induite par les serveurs. Bien qu'un tel indicateur puisse être manipulé, par exemple en augmentant la consommation des serveurs, il est actuellement l'indicateur privilégié de l'efficacité énergétique d'un centre. L'objectif d'un administrateur de centre est de choisir son matériel et son architecture de manière à s'approcher au plus près d'un PUE de 1. Cet indicateur est passé d'une moyenne de 2.5 en 2007 à 1.9 en 2011 [Sta13], tandis que GOOGLE déclare atteindre un PUE de 1.12 et Facebook un PUE de 1.09. De telles valeurs montrent l'engouement industriel pour la maîtrise énergétique des centres de données.

Une fois le matériel installé, la maîtrise énergétique du centre passe par l'observation et l'optimisation de l'exécution des services sur les serveurs.

2.2.4 Leviers logiciels

La maîtrise énergétique d'un centre nécessite de contrôler la consommation des serveurs. Différents moyens d'actions sont possibles, tant au niveau du contrôle de l'architecture physique qu'au niveau de la gestion des logiciels installés. Nous nous intéressons dans nos travaux à l'utilisation des leviers logiciels pour arriver à ces fins.

Approche au sein du serveur

Au niveau d'un serveur, l'administrateur peut utiliser les stratégies de DVFS pour réduire la consommation du CPU. Le DVFS consiste à adapter la puissance du CPU en fonction de sa charge. Cette adaptation peut être effectuée par la réduction de sa fréquence et la réduction du voltage en entrée de ses composants, ou par l'extinction de certains composants. Certaines fonctionnalités sont matérielles, telles le *clock gating* qui éteint certains composants du CPU lorsque les instructions en mémoire ne les nécessitent pas, ou logicielles, telles l'adaptation de fréquence ou la mise en veille des cœurs CPU. Ces stratégies permettent un compromis entre la réactivité du serveur à une hausse de charge CPU, et l'économie d'énergie induite par cette réduction de puissance.

Les autres composants d'un serveur peuvent aussi être partiellement éteints. Ainsi, une fois qu'un serveur est démarré, il est envisageable de mettre ses disques durs en veille. Une telle veille augmente le temps d'accès de ce serveur à ces disques, et doit donc être contrôlée logiciellement pour ne pas induire de latence. De même, en cas de présence de multiples cartes réseau sur le serveur, certaines peuvent être éteintes lorsque la charge réseau est faible. Les cartes GPU dédiées à certaines instruction (calcul massivement parallèle) sont aussi source de consommation inutile lorsque les programmes exécutés ne s'en servent pas, et peuvent donc être éteintes.

Finalement, lorsque le serveur n'est pas du tout utilisé, il devient intéressant de l'éteindre, et de ne l'allumer qu'au besoin.

Approche au niveau du serveur

Les serveurs modernes disposent d'un carte de gestion à distance, c'est-à-dire d'un circuit dédié. Celui-ci permet non seulement d'observer les activités, mais aussi d'éteindre et démarrer le serveur à distance, avec plusieurs niveaux d'extinction du serveur.

L'extinction traditionnelle du serveur est la méthode la plus simple, mais demande au serveur d'arrêter toutes ses couches logicielles. De plus, cette méthode nécessite le chargement de toute la couche logicielle (OS, pilotes, réseau) du serveur lorsque celui-ci doit être démarré.

La suspension sur disque du serveur consiste en l'enregistrement de son image mémoire sur le disque, puis en son extinction. Lors d'un éveil sur disque, le serveur démarre, puis recharge son image mémoire, et reprend son exécution là où il était avant suspension.

Le dernier cas d'arrêt du serveur consiste à suspendre son fonctionnement sans pour autant vider la mémoire. Cette suspension sur RAM a l'avantage de ne pas nécessiter de temps de chargement au démarrage, mais en contrepartie nécessite l'alimentation de la mémoire du serveur durant sa suspension, au contraire des deux méthodes précédentes qui éteignaient complètement le serveur.

Si l'arrêt des serveurs inactifs permet de réduire la consommation du centre, elle

pose cependant un problème de réactivité du centre. En effet, lorsque l'activité dans le centre augmente, il devient nécessaire de démarrer des serveurs suppléémentaires, ce qui nécessite un certain temps. L'administrateur d'un centre doit donc prévoir les pics de charge et s'assurer que le nombre de serveurs actifs soit suffisant pour amortir ces pics d'activité.

Cette approche par extinction des serveurs inactifs est encore améliorée par une allocation intelligente des tâches dans le centre.

Approche par contrôle de la charge

La consommation du centre peut être réduite par un placement intelligent des tâches sur les serveurs. Les tâches des centres peuvent être classées dans deux catégories : celles de type HPC aux activités très importantes et limitées par les capacités des serveurs, et celles de type webserver aux activités faibles mais variables. Contrôler la charge du centre demande de placer les VM sur les serveurs les plus efficaces, en prenant en cosidération les modèles des serveurs et les activités de ces serveurs.

Cette notion d'efficacité des serveurs dépend des modèles de serveurs du centre, cependant dans un centre homogène comme c'est le cas des centres HPC, il y a peu (souvent un seul) de modèles différents. Dans ce cas cette notion d'efficacité peut être déduite d'autres paramètres, tels la position des serveurs sur une armoire contenant déjà des serveurs actifs, la proximité de ces serveurs avec les équipements réseau ou la répartition des onduleurs du centre. En plus de la sélection des serveurs dédiés à l'exécution des tâches, l'administrateur d'un centre HPC peut utiliser la planification temporelle des tâches. Celle-ci permet de délayer certaines tâches afin de réduire la charge maximale du centre, et donc ne pas utiliser les serveurs les moins efficaces.

Dans les centres de type webserver, la charge d'une tâche est d'une part fortement variable, d'autre part inconnue au démarrage de cette charge. De plus, il n'est pas possible de reporter l'exécution des services, ces techniques ne sont alors pas applicables. Dans ces centres, il faut démarrer les tâches au plus tôt, puis s'appuyer sur le mécanisme de migration de la virtualisation pour ré-allouer les tâches.

Cette approche par sélection des serveurs hébergeurs peut être étendue dans les centres virtualisés par la possibilité de migrer dynamiquement les VM.

2.2.5 Gestion de l'énergie dans les systèmes virtualisés

Comme expliqué dans [BGDG+10], les systèmes de cloud computing permettent des manipulations spécifiques pour la maîtrise énergétique. Nous proposons ici quelques travaux liés à ces manipulations, en commençant par la modélisation des consommations des serveurs.

Consommation des serveurs

La maîtrise énergétique des centres est permise tout d'abord par la modélisation des consommations des serveurs. Une quantité importante de travaux ont été réalisés sur ce sujet, chaque nouvelle technologie de serveurs demandant de nouveaux travaux de modélisation. Il est ainsi possible d'après [Bel00] de définir précisément l'activité dans un serveur en utilisant les compteurs de performances, et de modéliser la relation entre l'activité d'un serveur et sa consommation. Différents méta-modèles de consommation sont comparés dans [RRK08].

En plus de l'activité du serveur, les technologies de gestion de l'énergie sont intégrées à des méta-modèles, telles le DVFS [HF05] dans [HM07] ou les états d'énergie des serveurs dans [GHBK11, GHBK12].

Le passage d'un centre physique à un centre virtualisé demande de mettre en relation la consommation d'un serveur non plus seulement avec son activité mais ausssi avec l'activité de ses VM, comme effectué dans [KAGS11] et [KZL+10]. La virtualisation permet alors de nouvelles techniques de gestion de l'énergie. Ainsi, [TYNW07] propose la gestion des états d'énergie dans un système virtualisé, tandis que [IMK+13] propose des niveaux d'énergie à faible latence et leur intégration dans les machines virtuelles.

Ces travaux de modélisation des consommations d'un centre ont permis de définir des algorithmes d'allocation des nouvelles tâches d'un centre prenant en compte sa consommation totale.

Placement des nouvelles tâches et consommation du centre

Le placement des nouvelles tâches, ou tâches entrantes, dans un centre a été d'abord effectué avec des algorithmes de type "first-fit", c'est à dire en attribuant les nouvelles tâches sur des serveurs dans un ordre prédéfini. Ces algorithmes ne considéraient aucun problème de consommation, aussi dès 2001 les problèmes de dissipation de la chaleur étaient évoqués [PBB+01] dans les centres HPC.

Dans ces centres de types HPC, le placement des tâches entrantes se fait sans connaissance de leur activité et ne peut donc pas en déduire d'effet sur la consommation des serveurs. Dans ce cas, les approches possibles sont des approches par à-priori réduisant la consommation du centre dans sa globalité. Ainsi [MCRS05] s'intéresse à la modélisation des points chauds dans un centre pour attribuer des priorités de placement des serveurs, produisant un algorithme que [BF07] évalue sur un centre réel. Par la suite, [TGV07] intègre la notion de système de recirculation de l'air dans un algorithme de placement dynamique des tâches. La priorité d'attribution des serveurs aux tâches devient ainsi dynamique, et après une phase d'observation se passe de sondes thermiques et énergétiques.

Dans le même temps, l'hétérogénéité des serveurs dans un centre, c'est à dire la présence de différents modèles de serveurs, donc de capacités (RAM, CPU) et de consom-

mations différentes, permet dans [SBP+05, NIG07] de déterminer des algorithmes de placement des tâches réduisant la consommation des serveurs.

Ces algorithmes permettent d'attribuer de nouvelles tâches dans un centre, d'autres travaux permettent d'améliorer l'état d'un centre en déplaçant les tâches dans celui-ci

Principes de la ré-allocation

Dans les centres traditionnels, l'allocation d'une tâche à un serveur est définitive. La virtualisation permet à l'administrateur de déplacer une VM déjà démarrée d'un serveur originel à un serveur cible, sans stopper l'exécution de cette VM.

La migration d'une VM permet d'augmenter les ressources disponibles pour celle-ci en la migrant vers un serveur ayant plus de capacités de calcul. Elle permet aussi, au contraire, de regrouper plusieurs VM sur un seul serveur. Ce dernier objectif, appelé aussi consolidation des serveurs, permet de réduire la consommation du centre en éteignant les serveurs peu utilisés.

Comme présenté dans [KBKK06], la migration d'une VM est soumise à des contraintes et l'administrateur doit prendre les bonnes décisions. Le principal soucis est d'éviter la surcharge des VM déjà présentes sur le serveur cible, on ne peut alors migrer une VM vers ce serveur que s'il dispose de suffisamment de mémoire et de capacité CPU libre pour l'accueillir. Si les activités CPU ou réseau du serveur sont trop importantes, la qualité d'exécution des VM qu'il héberge sera dégradée, par exemple avec des temps de réponse trop importants. Ces travaux présentent un algorithme de consolidation des serveurs, et d'amélioration de la qualité d'exécution des VM, par l'observation des activités des serveurs et de métriques de qualité des VM.

Si le framework proposé dans [VAN08a] permet de réduire la consommation d'un centre, il met aussi en exergue la difficulté de déterminer la consommation réelle d'un serveur dans un centre virtualisé. En particulier, il est difficile de déterminer le coût des actions de management. D'après les auteurs, il n'est cependant pas nécessaire d'avoir une connaissance totale de la consommation des serveurs. En effet la comparaison des efficacité de ceux-ci permet déjà d'obtenir des résultats probants. Ces travaux intègrent non seulement des modèles d'utilisation des ressources, mais aussi de consommation électrique, de rupture de contrats de service et de coût des migrations dans un framework à portée générique. Ils montrent ainsi l'intérêt d'avoir un système modulaire, où différents problèmes peuvent être considérés conjointement.

Les modèles plus complexes d'utilisation et de réservation du CPU par les VM utilisés dans des systèmes industriels sont pris en compte dans [CKS09]. Ces travaux proposent plusieurs algorithmes de gestion des VM, dans lesquels la consommation des serveurs est constante si le serveur est allumé et les serveurs sont triés par efficacité énergétique en CPU.

Dans le contexte HPC, [VAN08b] propose de migrer les VM en considérant des modèles de consommation des serveurs. D'après ces travaux, la consommation d'un

serveur est liée à sa charge CPU, il est donc possible de mettre en relation consommation du centre et charges CPU des serveurs. Cependant la couche de virtualisation entre les VM et le serveur physique implique une surcharge CPU, surcharge d'autant plus importante que les VM accèdent à la mémoire. Ces travaux nécessitent donc des schémas d'activité des VM du centre, des modèles de surcharge de virtualisation, et la connaissance des caractéristiques logicielles et électriques des serveurs. Ils ne sont donc pas utilisables dans le cas de gestionnaires de IaaS qui ne connaissent pas les relations entre VM.

Enfin, les travaux présentés dans [VBG09] considèrent non seulement la consommation des serveurs, mais aussi du système de refroidissement grâce à un modèle de re-circulation d'air. Ils proposent ainsi des algorithmes pour réduire la surcharge de refroidissement dans le centre due à cette re-circulation. Différents modèles de consommation des serveurs et climatisations sont utilisés, indiquant ainsi que l'amélioration d'un centre réel passe par l'analyse et l'adaptation des modèles aux observations.

Limites

À notre connaissance, aucune solution ne permet de faire de la gestion modulaire d'un centre virtualisé, bien que des travaux se soient essayés à une approche multi-critères. De plus, beaucoup de travaux se focalisent sur des centres homogènes en VM ou en serveurs, ce qui a du sens dans un centre de type HPC mais peu dans un centre virtualisé réel hébergeant des VM aux activités disparates.

2.3 Conclusion

Le cloud computing est une tendance majeure de l'informatique industrielle, liée à l'externalisation des services. Pour fonctionner, ce cloud computing nécessite des infrastructure techniques composées de milliers de machines. Ces milliers de machines consomment de l'énergie et posent à la fois un problème technique et éthique important et intéressant.

De ce fait, de nombreux travaux se sont focalisés sur la problématique de réduction de ce coût énergétique. Ces travaux sont très divers, de l'optimisation matérielle (serveur, switch etc.) à la gestion fine de la puissance processeur (DVFS). Dans le cadre de cette thèse, nous cherchons des solutions à cette problématique importante en nous focalisant sur le placement dynamique de tâche, sans arrêt de service, via la migration des environnements virtualisés.

Comme détaillé dans ce chapitre de nombreux travaux ont déjà eu lieu sur cette thématique du placement des tâches. Notamment, ceux effectués dans l'équipe d'accueil de ce travail de thèse, qui ont donné naissance au prototype de recherche ENTROPY et

sont présentés dans le chapitre suivant. L'objectif de cette thèse et la poursuite de ce travail en permettant de rendre ce gestionnaire plus performant et plus polyvalent.

3

Entropy : une première solution

Sommaire

ENTROPY a été la solution de Fabien Hermenier et Jean-Marc Menaud pour la gestion automatique du placement des VM d'un centre virtualisé. Cette solution, développée à partir de 2006, s'appuie sur un solveur de contraintes pour déterminer, de manière périodique, les actions à entreprendre pour améliorer l'efficacité du centre virtualisé. Le fonctionnement d'ENTROPY est donc basé sur la PPC.

3.1 Programmation par contraintes

La PPC est une méthode de résolution de problèmes, mathématiques et définis formellement, grâce à un programme d'exploration des solutions. Elle consiste en l'utilisation d'un langage de modélisation pour décrire un problème, puis en l'utilisation d'un programme, appelé *solveur*, pour trouver les solutions au problème ainsi décrit. Contrairement à des approches analytiques, telle la programmation linéaire, la PPC explore l'espace des variables du problème pour déterminer des solutions satisfaisant les contraintes.

La tâche d'un développeur utilisant la PPC consiste premièrement à exprimer problème dans le langage du solveur. Lorsque le solveur ne connaît pas les contraintes nécessaires ce problème, le développeur doit aussi développer ces contraintes. Enfin, les performances de résolution d'un problème de PPC sont très dépendantes des aides apportées par le développeur au solveur. Le développeur doit donc ajouter des aides de résolution dans le solveur.

3.1.1 Modélisation à base de contraintes

Le *modèle* d'un problème est défini par trois ensembles : un ensemble de variables, un ensemble de domaines de ces variables, et un ensemble de contraintes mettant en relation ces variables.

Les *variables* représentent les inconnues du problème, par exemple $x,\ y$. Elles peuvent être nécessaires à la description du problème, être utilisées pour des calculs intermédiaires, ou finalement être des valeurs constantes intégrées dans le problème. Par exemple, l'expression formelle $max(x, 3) + x$ nécessite au moins trois variables pour être modélisée : la variable nécessaire x, la constante 3 et la variable intermédiaire $max(x, 3)$.

Une variable possède un *domaine* qui est l'ensemble des valeurs que cette variable peut prendre, *eg.* $x \in [1..100]$ ou $y \in \mathbb{N}$. Quand le domaine d'une variable ne contient plus qu'une valeur, *eg.* $x \in [50..50]$, alors la variable est dite *instanciée*. Le domaine du problème est l'ensemble des domaines associés aux variables du problème. Une *évaluation* d'un problème est un ensemble de valeurs associées aux variable, par exemple $\{x = 1, y = 1\}$. À l'inverse, si le domaine d'une variable est vide, le problème est dit *incohérent*.

Les *contraintes* du problème représentent des relations entre les domaines des variables, *eg.* $x + y \leq z$ ou $(x \neq y) \Rightarrow (x + y = 5)$. Une contrainte est *satisfaite* dans une évaluation si sa relation est vérifiée par les valeurs des variables, *eg.* $x == y$ est satisfaite pour $x = 1, y = 1$. Cette contrainte est au contraire insatisfaite quand il n'est pas possible d'extraire des domaines de ces variables une instanciation satisfaisante. Par exemple $x + y = 0$ est satisfaite pour l'évaluation $\{x = 0, y = 0\}$ mais insatisfaite pour $x \in \{0, 1\}, y = 1$ ou $\{x = 0, y = 1\}$.

Un problème est défini par l'ensemble des domaines de ses variables. Lorsque l'on réduit le domaine d'une ou plusieurs variables du problème, on obtient un *sous-problème*. Les solutions d'un sous-problème sont donc aussi des solutions du problème père. Lorsqu'on travaille sur des entiers naturels ($\mathbb{N}$), les domaines des variables étant de taille infinie le nombre de sous-problème est aussi infini. En limitant les valeurs entières, par exemple en utilisant des entiers signés sur 32 bits, le nombre de sous-problèmes est fini. Celui-ci est cependant exponentiel avec le nombre de variables du problèmes.

Une *solution* d'un problème est une évaluation de ce problème pour laquelle toutes les contraintes sont satisfaites. Par exemple, un problème défini par la variable $x \in \mathbb{N}$ et la contrainte $x^2 + x = 0$ accepte $\{x = 0\}$ comme solution. De même, si les variables sont x, y, les domaines $(x, y) \in \{0, 1\}^2$ et les contraintes $x + y = 1, x \neq y$, une solution peut être $\{x = 0, y = 1\}$.

Par la suite, nous définissons les deux problèmes 3.1 et 3.2 qui serviront d'exemples. Le premier problème considère trois variables x, y, z dont les deux premières peuvent valoir 0 ou 1 et la troisième 1 ou 2. Les quatre contraintes spécifient que x est inférieur à y, lui même inférieur à z, que x est différent de z, puis que la somme de x, de y et de z vaut 3. Une solution de ce problème est $(x, y, z) = (0, 1, 2)$.

$$
\begin{aligned}
\text{v}ariables \quad & \begin{cases} (x, y, z) \end{cases} \\
\text{d}omaines \quad & \begin{cases} x \in \{0, 1\} \\ y \in \{0, 1\} \\ z \in \{1, 2\} \end{cases} \\
\text{c}ontraintes \quad & \begin{cases} x \leq y \\ y \leq z \\ x \neq z \\ x + y + z = 3 \end{cases}
\end{aligned}
$$

FIGURE 3.1 – Problème 1 de PPC.

Le deuxième problème considère trois variables a, b et c, chacune à valeur dans

les entiers naturels, ainsi que trois contraintes spécifiant que a et b sont égales, que la somme de a et b est égale à c, et finalement que la valeur de c multipliée par 2 vaut 8. Ce problème admet pour solution $(a, b, c) = (2, 2, 4)$.

$$
\begin{array}{ll}
\text{v}ariables & \left\{ (a, b, c) \right. \\
\text{d}omaines & \begin{cases} a \in \mathbb{N} \\ b \in \mathbb{N} \\ c \in \mathbb{N} \end{cases} \\
\text{c}ontraintes & \begin{cases} a = b \\ a + b = c \\ c \times 2 = 8 \end{cases}
\end{array}
$$

FIGURE 3.2 – Problème 2 de PPC.

Une fois le modèle défini, l'utilisateur utilise le solveur pour déterminer les solutions correspondantes.

3.1.2 Objectifs de résolution

Un problème modélisé par PPC peut avoir 0, 1 ou plusieurs solutions. Le solveur peut être paramétré pour démontrer différentes propriétés d'un problème.

Le solveur peut d'abord déterminer la faisabilité du problème, c'est-à-dire que celui-ci admet bien au moins une solution. Cette existence est prouvée en exhibant une solution au problème. L'objectif de la résolution sera alors de trouver au plus vite une instanciation du problème qui satisfasse toutes les contraintes.

Le solveur peut, au contraire, déterminer qu'un problème est insolvable, c'est-à-dire qu'aucune instanciation n'est satisfaisante. Cette démonstration implique un parcours partiel des sous-problèmes et des démonstrations formelles d'insatisfaction. L'objectif du solveur est alors de décomposer le plus rapidement possible le problème en un ensemble de sous-problèmes pour lesquels les contraintes permettent de déduire une incohérence de manière formelle.

Enfin, le solveur peut être utilisé pour déterminer non pas une solution mais la meilleure solution possible au problème, selon un critère d'évaluation. L'objectif est alors, d'une part d'exhiber une bonne solution, et d'autre part de montrer que ce problème n'admet aucune meilleure solution que celle-ci.

Différentes techniques permettent de déterminer rapidement les solutions, ou l'absence de solution, d'un problème. Nous allons détailler les principes de résolution dans la PPC et certaines de ces techniques.

3.1.3 Exploration d'un arbre des solutions

La manière "naïve" de résoudre un problème consiste à énumérer toutes ses instanciations, en vérifiant pour chacune si les contraintes sont satisfaites. Étant donné que les solutions des sous-problèmes sont aussi solutions du nœud père, il est possible de décomposer le problème en sous-problèmes de domaines réduits dans lesquels chercher les solutions. Une approche de cette décomposition consiste par exemple à sélectionner une variable, puis à générer un sous-problème pour chaque valeur de son domaine. Cette approche est récursive, puisqu'elle peut être réutilisée pour les sous-domaine ainsi générés.

Pour effectuer une telle exploration, le solveur construit récursivement un *arbre de parcours* à partir du problème initial, qui en est la racine, où chaque nœud de l'arbre correspond à un sous-problème de son nœud père. Un nœud dont toutes les variables sont instanciées ne peut avoir de fils et est donc une feuille, qui représente une solution potentielle au problème. Lors d'une telle exploration, il est indispensable que passage d'un nœud père à ses fils conserve les solutions possibles. Pour cela, le domaine du nœud père doit être contenu dans l'union des domaines des nœuds fils, ce qui est le cas par exemple lors de l'énumération des valeurs d'une variable.

Nous proposons dans la figure 3.3 un arbre d'exploration possible de problème 1(3.1). Cet arbre est obtenu en énumérant les valeurs pour la variable x, puis pour la variable y et enfin pour z.

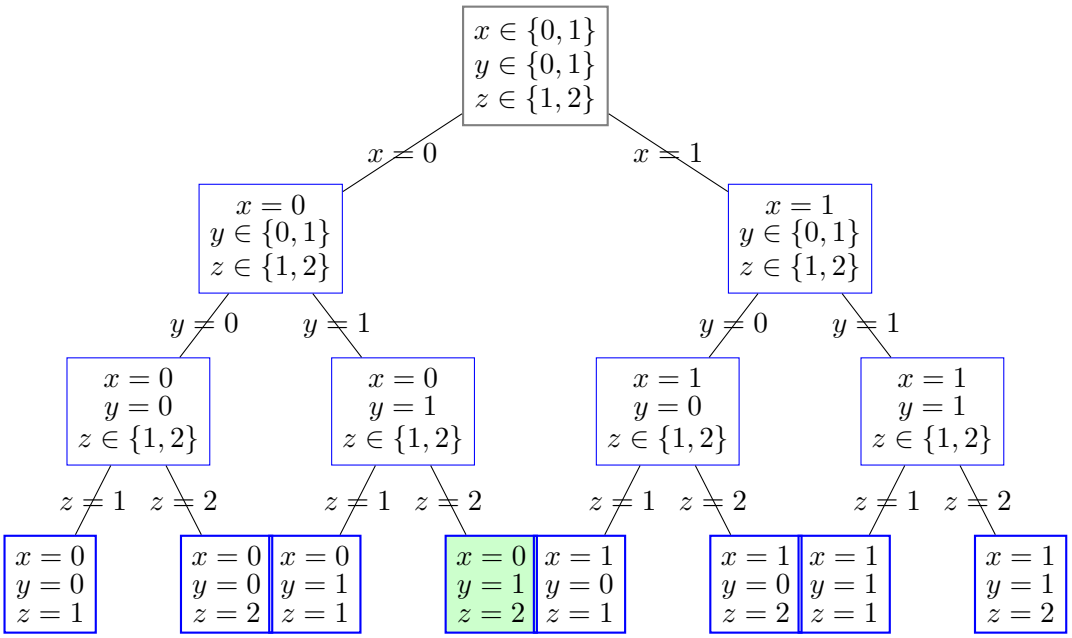


FIGURE 3.3 – Exemple d'abre des solutions d'un problème.

Durant la construction de cet arbre, les feuilles obtenues sont évaluées et seules celles satisfaisant toutes les contraintes sont retenues. Dans l'exemple, seule la solution $\{(x, y, z) = (0, 1, 2)\}$ satisfait toutes les contraintes et est donc la seule solution que le solveur retient. Cet arbre est construit par l'instanciation à chaque nœud fils d'une variable non instanciée dans le nœud père. Une seule variable étant sélectionnée à chaque nœud père, un nœud père a ici autant de fils que la taille du domaine de la variable à instancier. Cette instanciation d'une seule variable n'est pas systématique et dépend des stratégies d'exploration de l'arbre.

Ce problème de 3 variables, chacune avec deux valeurs possibles, a donc $2^3 = 8$ solutions potentielles. L'exploration a demandé ici 6 nœuds intermédiaires, alors qu'une seule solution était présente. Cette exploration naïve a donc demandé l'exploration de 7 solutions inutiles, et 4 nœuds intermédiaires sans solutions. Cette approche naïve, si elle permet de comprendre le problème, souffre cependant de très mauvaises performances.

Les performances d'une recherche d'une solution sont liées à la stratégie d'exploration. En l'occurrence, connaissant déjà une solution, une première stratégie est d'affecter directement cette solution au domaine du problème. Ainsi, après création d'un seul nœud $\{(x, y, z) = (0, 1, 2)\}$, le solveur peut déterminer qu'au moins une solution existe, c'est-à-dire que le problème est *décidable*. Cependant cette stratégie n'aurait aucun effet si le problème n'avait pas de solution, ou qu'il fallait énumérer toutes ces solutions.

Si cette énumération est efficace pour des problèmes simples, tel celui donné en exemple, son temps de résolution est proportionnel au nombre de feuilles de l'arbre. Ce nombre est égal au produit des tailles des domaines des variables, dans l'exemple du problème 1(3.1) ce produit vaut $2 \times 2 \times 2 = 8$ feuilles. Si les domaines avaient été $x, y, z \in \mathbb{N}$, les solutions auraient été les mêmes, mais l'arbre de recherche aurait eu une taille infinie, rendant cette énumération impossible.

Ainsi le problème 2(3.2), bien que trivial pour un humain, n'est pas résoluble pas un tel algorithme. Il est donc important de réduire dès que possible la taille de l'arbre de recherche.

3.1.4 Cohérence des nœuds intermédiaires

Lors de la génération de l'arbre de recherche, certains nœuds sont incohérents, c'est à dire qu'au moins une contrainte ne peut plus être satisfaite. Dans notre exemple, le nœud $\{x = 1, y = 0, z \in \{1, 2\}\}$ est en contradiction avec la contrainte $x \leq y$. Il n'est donc pas nécessaire d'explorer les fils de ce nœud car ceux-ci seront forcément incohérents. Détecter ces incohérences au plus tôt permet d'élaguer l'arbre de recherche, et donc de réduire le coût d'exploration de cet arbre.

La détection de ces incohérences est effectuée à chaque transition d'un nœud père vers un nœud fils. Elle est donc assurée de manière récursive, en commençant par vérifier si les contraintes sont cohérentes à la racine. Dès que le domaine d'une variable est modifié, par exemple au passage d'un nœud père à son fils par l'instanciation de

cette variable, il faut vérifier toutes les contraintes liées à cette variable. Quand une de ces contraintes s'avère non satisfaite, le nœud fils est déclaré incohérent et il est inutile de continuer son exploration, le solveur reprend alors l'exploration du nœud père. Cette méthode de mémorisation des nœuds et de retour au nœud père est appelée *backtracking* puisqu'elle remonte l'exploration de l'arbre des solutions en considérant le nœud actuel totalement exploré.

Avec le backtracking, l'arbre de recherche précédent devient 3.4 :

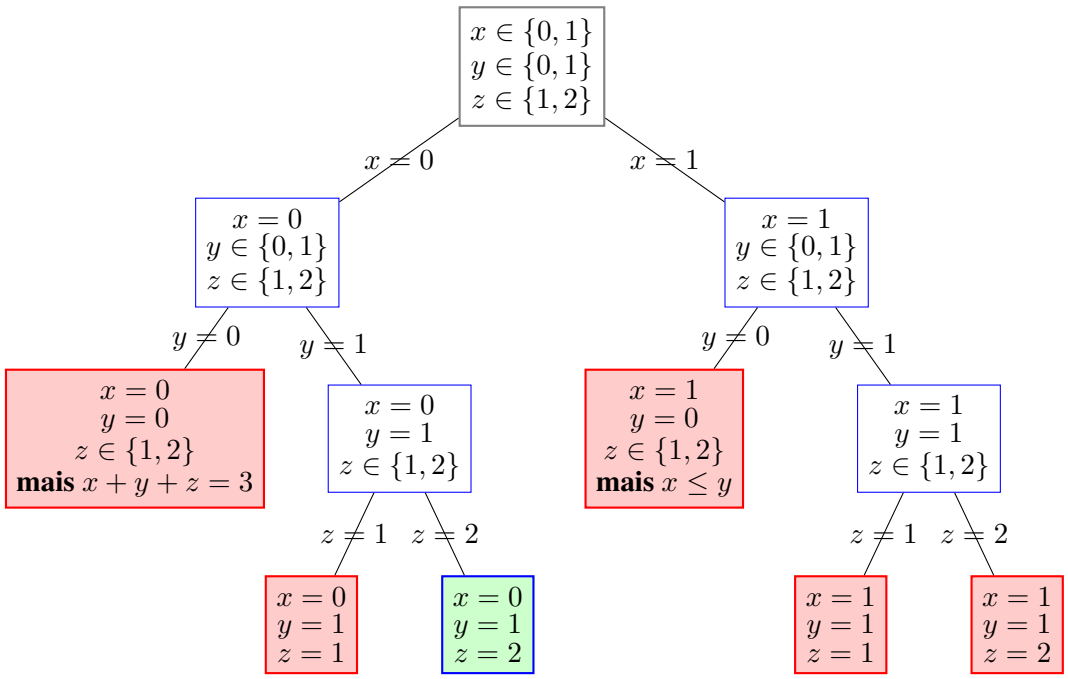


FIGURE 3.4 – Arbre des solutions avec backtrack.

Bien que le backtracking permette de détecter au plus tôt les nœuds incohérents, il peut être intéressant de réduire le domaine d'une variable en fonction des contraintes auxquelles elle participe.

3.1.5 Filtrage des contraintes

En utilisant la définition formelle des contraintes, il est possible d'exprimer le domaine d'une variable en fonction des domaines des autres variables participant à cette contrainte. Ainsi, si $x \in [1, 2]$ et la contrainte $y = x$, je peux trivialement en déduire $y \in [1, 2]$. Le filtrage consiste à appliquer une contrainte pour réduire le domaine des variables y participant. Le solveur restreint alors le domaine d'une ou plusieurs variables d'un nœud à partir des contraintes et des domaines des variables de ce nœud. Cette

propagation au niveau d'un nœud des domaines des variables grâce aux contraintes est appelée le *filtrage* des domaines.

L'opération de filtrage sur un nœud est réalisée en une ou plusieurs passes : si une première passe réduit le domaine d'une variable, il est possible qu'une passe ultérieure réduise encore des domaines de variables. Si après un ou plusieurs filtrage le domaine d'une variable est vide, alors le nœud est incohérent et ne peut donc avoir de solution.

Par exemple, dans problème 2(3.2), si $a = 1$, alors d'après la contrainte $a = b$ on a $b = 1$. On peut donc filtrer les valeurs de b différentes de 1. Mieux encore, dès la racine la contrainte $2 \times c = 8$ peut être filtrée en $c = 4$. Une deuxième passe de filtrage utilise cette fois la contrainte $a + b = c$ pour en réduire l'espace de a et de b. En effet, $a + b = c$ implique alors $a = 4 - b$ et $b = 4 - a$, après filtrage $a \in [0..4], b \in [0..4]$. Avec le filtrage, le domaine des variables de ce problème passe alors de $(a, b, c) \in \mathbb{N}^3$ à $(a, b) \in [0..4]^2, c = 4$. Le domaine de chacune des variables étant alors fini, ce problème devient résoluble en explorant l'arbre de recherche comme le montre la figure 3.5.

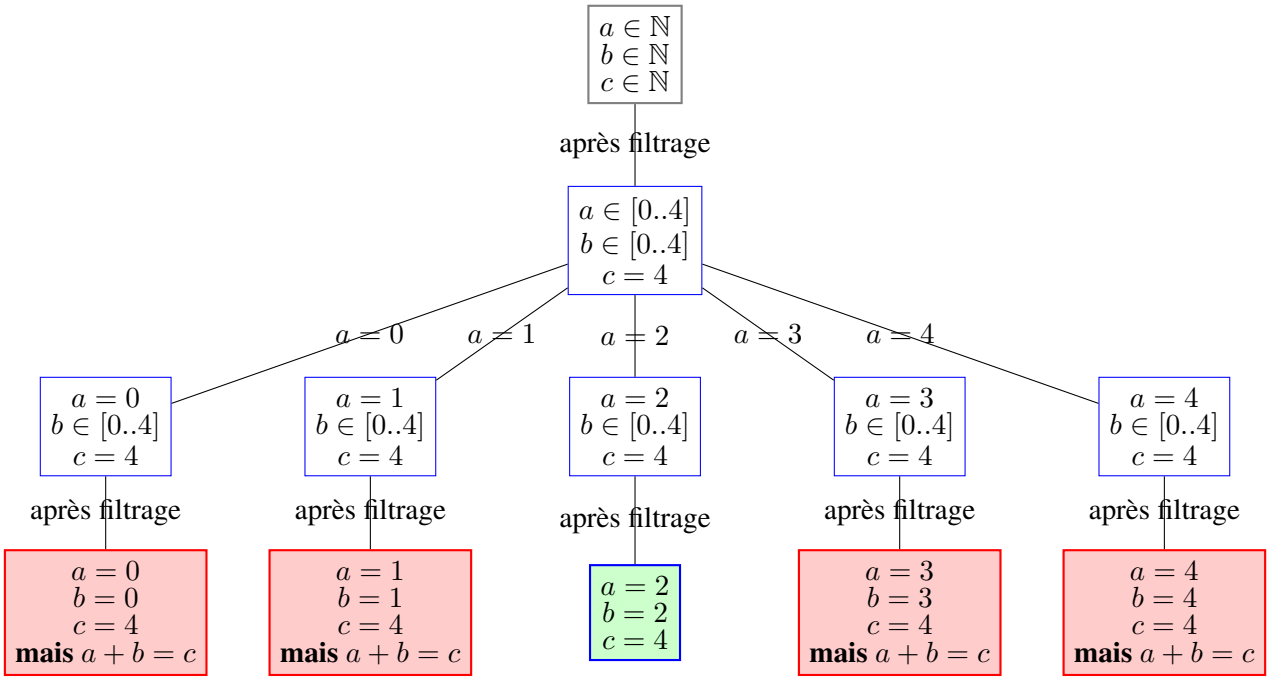


FIGURE 3.5 – Arbre des solutions avec filtrage.

Le filtrage des contraintes peut être une fonctionnalité très coûteuse à mettre en place. En effet elle demande de déduire pour chaque contrainte des restrictions sur le domaine de chaque variable, demandant un temps de calcul non négligeable. Un filtrage trop poussé peut donc ralentir le processus de résolution dans le cas de contraintes com-

plexes. Il est alors nécessaire de trouver un équilibre entre le temps de construction de l'arbre de recherche et le temps passé à filtrer.

Le solveur peut aussi être utilisé pour trouver la meilleure solution au problème. Dans ce cas, il ne s'arrête pas à la première solution et continue à chercher d'autres meilleures solutions.

3.1.6 Optimisation et contraintes

La PPC permet de définir des variables et des relations sur ces variables, mais aussi une fonction d'évaluation des solutions du problème. La résolution d'un problème peut alors être paramétrée pour rechercher non pas la première solution, mais une solution optimale, c'est-à-dire maximisant ou minimisant cette fonction.

Un algorithme simple de recherche de solution optimale consiste à parcourir toutes les solutions du problème, à les évaluer et à ne conserver que la meilleure. La complexité de cet algorithme est la même que celle de l'algorithme de parcours exhaustif des solutions au problème.

Une amélioration de cet algorithme consiste à ajouter une contrainte supplémentaire d'évaluation au problème : celle ci impose que toute solution déterminée par la suite soit meilleure que celle déjà obtenue. Cette contrainte considère une variable extérieure au modèle, la meilleure évaluation déjà obtenue, qui n'a pas de sens dans le problème mais en a pour la résolution de celui-ci. Le filtrage de cette contrainte améliore ainsi le processus de résolution, élaguant les branches de l'arbre qui ne peuvent formellement amener une solution suffisamment bonne.

Cette variable étant modifiée à la découverte d'une meilleure solution, elle peut induire une incohérence depuis le nœud actuellement explorée jusqu'à un nœud très proche de la racine. Il est alors possible de redémarrer le processus de résolution du problème depuis la racine en conservant cette nouvelle valeur de solution optimale. Ce paramétrage réduit le coût du backtracking lorsque l'arbre de recherche a une profondeur importante.

Dans tous les cas, la recherche de la meilleure solution d'un problème demande une exploration plus profonde de l'arbre des solutions, et donc un temps de résolution plus important. Ainsi, les paramètres de filtrage et d'objectif ont une influence directe sur les performances du solveur. Ces performances dépendent aussi des stratégies de recherche lors de l'exploration de l'arbre des solution. Si des méta-stratégies génériques existent et permettent de résoudre certaines classes de problèmes, les stratégies les plus intéressantes sont généralement celles spécifiques au problème considéré.

3.1.7 Stratégie de recherche

La PPC permet de modéliser un problème avec des contraintes variées, en particulier grâce à la présence de catalogues de contraintes génériques. Elle a en contrepartie de

cette généricité un problème de performances. En effet, si la modélisation d'un problème peut être effectuée facilement, le parcours exhaustif d'un arbre de recherche peut être désastreux. Par exemple, si un problème possède 50 variables, chacune avec 1000 valeurs possibles, alors le problème peut être instancié de $1000^{50} = 10^{150}$ façons différentes. Le filtrage des contraintes permet éventuellement de réduire l'arbre des solutions, mais sa taille restera un problème pour un parcours exhaustif.

Si l'objectif est de trouver une solution au problème, il faut donc guider la construction de l'arbre de manière à arriver au plus tôt vers une telle solution. Si au contraire l'objectif est de démontrer l'absence de solution, il faut guider le solveur pour lui faire atteindre au plus tôt des nœuds inconsistants. Ce guidage est permis par l'adaptation du solveur au problème à résoudre. Cette adaptation passe par le développement de stratégies de recherche pour le solveur.

La création et l'évaluation d'une stratégie de recherche demande un travail important, si ce n'est le plus important.

Les heuristiques associées à un problème

Dans le cadre de la PPC, une heuristique est une stratégie d'exploration de l'arbre des solutions. Cette aide, apportée par une connaissance de la sémantique du problème ou par des résultats empiriques, détermine pour certains nœuds de l'arbre le nœud suivant à explorer.

Une heuristique indique au solveur comment se déplacer intelligemment dans l'espace des solutions. L'objectif est donc d'approcher la meilleure solution possible dans un délai de temps raisonnable. La recherche et la sélection d'heuristiques appropriées est une étape importante lors de l'utilisation de la PPC pour résoudre un problème. Une heuristique est dédiée à la résolution d'un problème, ou d'une classe de problèmes, et doit donc être développée spécifiquement pour chaque problème.

Les heuristiques développées dans le cadre d'un problème peuvent être comparées, en terme de performances, aux heuristiques de résolution génériques.

Les méta-heuristiques

Si des heuristiques très spécifiques peuvent améliorer un problème particulier, des heuristiques génériques sont aussi accessibles. Ceux-ci permettent de rechercher efficacement une solution optimale dans un problème sans nécessiter de développement important ou de connaissances particulières. Une telle heuristique est appelée *méta-heuristique*, dans le sens où elle produit pour un problème donné une heuristique de parcours. Si la production d'une heuristique considère la sémantique du problème qu'elle adresse, une méta-heuristique ne considère que la modélisation du problème.

Les principales méta-heuristiques utilisées pour résoudre des problèmes à variables discrètes sont :

- Le recuit simulé, qui explore l'espace de recherche tout en acceptant de dégrader sa solution afin de sortir des optimums locaux. Tout au long du processus, le recuit va de moins en moins accepter ces dégradations, ce qui va le faire converger vers un optimum (que l'on espère) global.

- la recherche avec tabous, qui a l'inverse du recuit simulé est déterministe et a une notion de mémoire. Le choix du meilleur voisin d'une solution pousse l'algorithme à trouver les optimums locaux. Comme l'exploration de l'espace de recherche est effectué en limitant le voisinage de la solution en rendant "tabous" certains mouvements, l'algorithme doit théoriquement visiter l'optimum global.

- les algorithmes évolutionnaires, issus de la théorie de l'évolution de Darwin, qui manipulent plusieurs solutions en même temps, et qui en les combinant forment de nouvelles solutions. Le fait d'avoir une population de solutions facilite l'exploration de l'espace de recherche, et les meilleures solutions seront favorisées pour participer à la création de nouvelles solutions ce qui aura pour effet de favoriser les combinaisons des "bonnes caractéristiques", et donc de trouver un optimum global.

Finalement, la PPC permet de combiner différentes approches durant la résolution d'un problème. Un développeur peut ainsi guider le solveur par l'exclusion des solutions qu'il sait peu satisfaisantes, puis utiliser une méta-heuristique pour évaluer la meilleure solution parmi le domaine restant.

3.1.8 Les solveurs de contraintes

Les différentes implémentations des solveurs ont chacune leurs propres caractéristiques.

Le langage d'implémentation du solveur conditionne le cadre de son utilisation. Certains langages sont plus performants que d'autres lors de l'exécution d'un algorithme d'exploration d'arbre. En effet, les langages dits de “haut niveau” intègrent des comportements génériques, laissés à la charge des développeurs dans les langages dits de plus “bas niveau”. Ces implémentations génériques réduisent le coût de développement des programmes, et apportent des solutions testées et donc à priori exemptes de bugs, mais ne permettent généralement pas d'atteindre des performances optimales, accessibles par des implémentations spécifiques au problème. Ainsi, le solveur Choco développé en Java sera moins performant pour le même parcours que le solveur Gecode développé en C++.

Certains solveurs proposent des fonctionnalités d'optimisation des problèmes à résoudre. Par exemple, la recherche de formes de symétrie dans le problème [BS07] permet de réduire grandement la taille de l'arbre de recherche. De même la déduction formelle de contraintes intermédiaires permet de filtrer plus rapidement le problème. Par exemple, des deux contraintes $x + y = c$ et $x = y$ on peut déduire les contraintes

$2x = c$ et $2y = c$. Cependant la recherche de telles optimisations put prendre un temps supérieur à celui qu'elle fera gagner, voire gaspiller un temps de calcul pour ne déduire aucune optimisation.

Certains solveurs peuvent interpréter des problèmes à partir de leur description dans un modèle générique, cependant différents solveurs peuvent proposer différents langages de description. Ainsi il n'est pas toujours possible de définir un problème de la même façon dans différents solveur, et donc certains problèmes ne peuvent pas être résolus par tous les solveurs.

En particulier, bien que le langage de description du problème puisse être indépendant du langage d'implémentation du solveur, le développement de contraintes spécifique est lié à ce dernier. Il est alors difficile de produire des contraintes pour tous les solveurs. Dans les faits, chaque solveur propose son ou ses catalogues de contraintes pré-définies, les spécifications d'une contrainte peuvent varier d'un catalogue à l'autre, et évidemment d'un solveur à l'autre.

Enfin, la capacité du langage à intégrer de nouvelles contraintes, spécifiques à une classe de problèmes, conditionne l'évolutivité de ce langage. Si un solveur est performant mais fermé à l'évolution des problèmes qu'il adresse, la PPC perd de son sens et peut parfois être remplacée par des méthodes de résolution analytiques.

Une équipe travaillant sur le solveur Choco à l'EMN nous apportant le support technique nécessaire, nous n'avons pas étudié tous les solveurs existants. De plus, ce solveur était déjà intégré à ENTROPY, nous nous sommes appuyés sur cette base déjà éprouvée.

3.2 Entropy

ENTROPY permet d'administrer un centre virtualisé de type IaaS par l'automatisation des actions d'administration. Ces actions incluent le démarrage et l'arrêt des serveur du centre, ainsi que les migration, suspension et reprise des VM, selon des règles spécifiées par l'administrateur. ENTROPY déduit et exécute de manière régulière un plan de reconfiguration du centre.

La version 1.0 d'ENTROPY avait pour objectif de regrouper automatiquement les VM sur un nombre réduit de nœuds afin de réduire la consommation énergétique du centre. Ce placement était effectué en considérant les besoins des VM en ressources CPU et mémoire, ressources proposées par les serveurs. Le problème adressé était alors un problème de satisfaction des contraintes de ressources dans le centre. En effet, il ne suffit pas de placer toutes les VM sur un seul serveur, puisque les ressources disponibles pour chaque VM peuvent être insuffisantes. Ce problème de satisfaction des contraintes de ressource est un problème dit de *packing*. Ce problème est très difficile à résoudre, aussi ce travail a porté tant sur la modélisation d'un centre grâce à un solveur de contraintes que sur la recherche de solution optimale dans des centres de petite taille.

En plus du problème de packing des ressources, ce travail considérait un problème

de planification des tâches administratives. Chaque tâche d'administration proposée par ENTROPY y nécessite une durée d'exécution. Durant cette durée, la tâche peut empêcher l'exécution d'autres tâches sur les serveurs concernés par cette tâche. De ce fait, une solution au problème de regroupement des VM peut être très lente à mettre en place. Ce système de planification permettait de réduire le temps nécessaire à l'exécution d'un plan de reconfiguration proposé par ENTROPY. De plus, elle permettait d'éviter des problèmes d'interblocage de règles d'administration, par exemple lorsqu'ENTROPY propose l'échange de deux VM de deux serveurs différents à forte activité CPU.

La version 2.0 d'ENTROPY remplaçait ce problème initial par des règles de placement des VM. Par exemple, l'administrateur peut demander le placement de plusieurs VM sur le même serveur, ou au contraire leur répartition sur différents serveurs. Dès lors, il ne s'agit plus d'un problème de regroupement des VM, mais de satisfaction des règles de l'administrateur dans un problème fortement contraint. Dans cette version, ENTROPY ne cherche pas à réduire le nombre de serveurs mais à satisfaire les contraintes en proposant le plan de reconfiguration le plus court possible.

ENTROPY utilise le système de supervision Ganglia[MCC04] pour l'obtention de la configuration du centre virtualisé et la bibliothèque de spécification et résolution de contraintes Choco[cho] pour calculer un placement et planifier le ré-agencement. Grâce à la PPC, ENTROPY permet d'appréhender des problèmes d'administration d'un centre virtualisé, qui peuvent nécessiter l'expression de contraintes complexes.

3.2.1 Boucle autonomique

ENTROPY fonctionne sur le principe d'une boucle de contrôle, qui effectue plusieurs opérations successives de manière répétitive. Cette boucle alterne les phases d'observation du centre, d'analyse, puis d'exécution d'actions pour optimiser la configuration du centre.

Tout d'abord, ENTROPY utilise un système de monitoring pour obtenir des informations sur le centre virtualisé. Ce système lui permet de déterminer la configuration virtuelle, et en particulier les utilisations des CPU et RAM des VM.

Une fois l'état du centre virtualisé connu, ENTROPY déduit, à partir de cet état et des règles de l'administrateur, les variables et contraintes définissant le problème dans Choco. Chaque version d'ENTROPY paramètre de plus ce problème avec ses propres objectifs d'optimisation et ses stratégies de recherches. Quand toutes les contraintes ont été ajoutées, ENTROPY utilise le solveur de PPC pour déterminer une configuration viable et respectant les objectifs de l'administrateur.

ENTROPY utilise le solveur de PPC pour déterminer non seulement une configuration cible, mais aussi un plan de reconfiguration du centre virtualisé. Ce plan est constitué d'une succession d'actions d'administration sur le centre virtualisé amenant celui ci à la configuration cible. Il peut être déduit de la configuration à atteindre ou bien modélisé dans le solveur de contraintes.

Ce plan de reconfiguration déterminé, ENTROPY exécute la séquence d'actions qui le compose, et débute une nouvelle boucle de reconfiguration.

3.2.2 d'Entropy à OPTIPLACE

ENTROPY est un logiciel de gestion autonome, effectuant à la fois la récupération des informations du centre, la prise en compte des demandes de l'administrateur, la résolution du problème de reconfiguration, et l'exécution du plan de reconfiguration. Cependant, l'évolution des fonctionnalités dans ces domaines a induit une complexité importante et nous a amené à nous concentrer sur la résolution du problème.

D'une part, les système d'observation de centre, considérés initialement comme de simples moniteurs centralisés, ont évolué vers une utilisation industrielle et pour cela intégré des fonctionnalités supplémentaires. Ainsi, la solution fournie par VMWare incorpore les notions de compte utilisateur, de droit d'accès aux données, ainsi que des systèmes de surveillance événementielle. De même, des fonctionnalités de gestion complexes peuvent être présentes dans les hyperviseurs. L'hyperviseur Xen par exemple permet à l'administrateur du centre de spécifier des priorités d'accès aux ressources. De ce fait, la compréhension et l'observation d'un centre réel demande l'utilisation de pilotes spécifiques à ce centre, et donc un surcoût important de développement.

D'autre part, les fonctionnalités d'interaction nécessaires à l'optimisation d'un centre ont suivi cette même tendance de complexification. Certains centres permettent par exemple de migrer une VM par suppression de celle-ci et clonage d'un modèle de VM sur un autre serveur. Cette fonctionnalité permet de réduire le temps de migration nécessaire pour une VM sans mémoire, ou encore d'équilibrer les charges parmi plusieurs serveurs.

Ces systèmes de droits, de priorités, d'équivalence des VM doivent être pris en compte et intégrés à ENTROPY par le pilote du centre de manière transparente, ce qui demande un travail d'adaptation supplémentaire. Nos travaux ont donc transformé le programme autonome qu'était ENTROPY pour ne conserver que la fonctionnalité de résolution des problèmes, afin de l'intégrer à BTRCLOUD. Ceci fait, nous nous sommes attaché à repousser les limites ENTROPY telles qu'elles étaient présentes dans la version 2.0.

3.3 Limites d'Entropy 2.0

ENTROPY était dédié à la résolution de problèmes de packing et de satisfaction de contraintes de placement. Notre travail portait essentiellement sur l'amélioration du système de parking interne et sur son ouverture à des contraintes de placement nouvelles. Cependant, son utilisation dans des cadres scientifiques différents nécessite des adaptations complexes.

3.3.1 Approche tout en un

ENTROPY fonctionnait sur la base de l'autonomic computing, du monitoring jusqu'au système d'exécution. Nous avons repensé l'architecture (nommé btrCloud) en vue de la rendre plus modulaire et agnostique du système de IaaS sous-jacent, produisant ainsi OPTIPLACE. OPTIPLACE n'est plus que la brique d'aide à la décision pour la gestion d'un centre virtualisé, les observations, analyses, modélisations, étant faites en amont et l'exécution des décisions étant effectuée en aval.

3.3.2 Approche non flexible

La version 2.0 n'intégrait plus de système d'optimisation. Pendant nos premiers travaux sur l'intégration d'un nouveau système d'optimisation énergétique, le système ne fournissait plus de réponse dans des temps acceptables. Cette perte de performance était liée à un système de planification coûteux et des heuristiques trop agressives, développés dans le cadre du placement de tâche sous contraintes. Le système de planification ajoutait un nombre important de contraintes dan le solveur, tandis que les heuristiques élaguaient l'arbre de recherche trop rapidement, si bien que le système d'optimisation ne pouvait plus fonctionner.

Comprendre un tel système, l'adapter et le corriger est une tâche bien trop complexe pour un non initié. Notre objectif dans OPTIPLACE est de fournir une solution flexible permettant à tout développeur Java de développer son propre système de placement et d'optimisation d'un centre virtualisé en lui fournissant les bonnes abstractions.

Ajout de contraintes difficiles

L'ajout de contraintes autour de la notion de VM/serveurs reste cependant difficile. Elle impose de connaître le système dans sa totalité et nécessite la manipulation des domaines de variables dans le solveur. Cette tâche reste complexe, et peut dans certain cas interférer avec des objectifs d'optimisation. Un travail important dans OPTIPLACE a été effectué pour faciliter cette tâche.

Fragmentation des versions

L'architecture d'ENTROPY lui permettait d'être plus efficace dans la résolution de problèmes, tous les paramètres étant connus lors du développement. Cet avantage se transforme en inconvénient lorsque différents développeurs veulent modifier le code d'ENTROPY, leurs objectifs pouvant être divergents.

Les modification d'ENTROPY ont chacune été effectuées dans leur propre version, développée comme un projet à partir d'une version fixée d'ENTROPY. Cette manière de développer apporte la stabilité nécessaire à la création de projets complexes, mais

empêche l'évolution simultanée des différents projets. En effet l'intégration de modifications apportées au projet initial ENTROPY (sous la forme de correctifs) doit être adaptée à chaque projet lié. Cette adaptation pouvant être très coûteuse en temps de développement, généralement les développeurs n'appliquent pas les correctifs apportés au projet initial.

Dès lors, chaque projet basé sur ENTROPY se contente d'utiliser une version fixée d'ENTROPY, dont il ne suivra plus les évolutions. Les optimisations de code, les corrections de bugs, les ajouts de fonctionnalités apportés par les travaux de Fabien Hermenier ne peuvent alors être intégrés à des projets anciens. Les différentes versions d'ENTROPY, les différents projets utilisant chacun sa version, tous optimisés pour un problème spécifique, ne peuvent plus bénéficier des apports apportés aux uns ou aux autres.

Redondance du code et des contraintes

Cette fragmentation a un deuxième effet sur le développement, la redondance du code. En effet chaque développeur devant intégrer un nouveau problème ne cherchera pas à utiliser du code d'un projet existant, celui-ci étant probablement lié à une version différente d'ENTROPY. Il doit ainsi re-coder des fonctionnalités dans un nouveau projet, sans pour autant y apporter d'amélioration, mais uniquement pour l'adapter à une nouvelle version d'ENTROPY. Ce code est donc redondant dans les projets basés sur ENTROPY et en particulier, les bugs présents sont redondants et, comme indiqué dans la section précédente, non corrigés.

Cette redondance de code entraîne à son tour une redondance des contraintes intégrées à un problème de placement. Ainsi, quand un développeur importe des classes d'un autre projet, ces classes peuvent créer des contraintes équivalentes à celles que le développeur crée dans son propre code.

Par exemple, la manipulation de groupes de VM n'est pas présente dans ENTROPY. Les contraintes assimilées, telles le décompte des VM de ce groupe hébergées par un serveur, intégrées parfois à des heuristiques, doivent être recodées pour chaque projet utilisant cette notion. Ainsi les mêmes classes peuvent être produites de différentes manières dans différents projets. Si ces classes sont utilisées dans un projet tierce, elle vont alors produire des contraintes équivalentes, et donc des doublons de contraintes.

En plus du problème d'ajout de contraintes redondantes dans le solveur, l'ajout de code équivalent multiplie les risques de bugs. Que ce soit à cause des difficultés à intégrer des contraintes externes, ou tout simplement par un besoin accru d'écriture de ligne de codes, le développeur qui intègre des contraintes extérieures aura mathématiquement plus de travail et donc d'erreurs dans son code.

Cette redondance peut être évitée par la mutualisation des fonctionnalités. Ainsi la création de modules développeurs, dédiés à l'implémentation de fonctionnalités spécifiques, permet cette mutualisation des travaux. Un développeur peut alors produire un

module, qui sera utilisé de la même manière par d'autres développeurs sans que ceux-ci aient besoin de modifier le code des fonctionnalités. Il est alors possible d'évaluer une nouvelle implémentation de fonctionnalité en remplaçant le module ancien par celui-ci.

Évaluation sur simulation

ENTROPY a été rarement évalué sur infrastructure réelle. Dans le cadre d'OPTIPLACE et de btrCloud, nous avons tenu à valider nos solutions sur infrastructure réelle. En effet, c'est après tests que nous avons remarqué qu'il était nécessaire de séquencialiser l'ensemble des migrations partant d'un même noeud et/ou arrivant sur un même noeud.

3.4 Conclusion

Nous voulons développer une nouvelle version, flexible et modulaire, en ajoutant une gestion de l'énergie. Nous voulons tester la solution sur infrastructure réelle. Le chapitre suivant décrit plus précisément l'architecture de btrCloud, et les limites d'Entropy.

Contribution

4

OptiPlace

Sommaire

Le point de départ de ces travaux sur l'optimisation énergétique fût d'assimiler les travaux antérieurs réalisés autour d'ENTROPY, en vue d'apporter de nouvelles contributions scientifiques. Comme précisé au chapitre 3, les premiers travaux réalisés par Fabien Hermenier [Her09] ont permis le développement d'ENTROPY (V1.0), un gestionnaire de centre. Son objectif était de permettre l'exécution de l'ensemble des machines virtuelles sur un minimum de serveur physique. Les serveurs physiques libérés par le gestionnaire peuvent être éteint ou réutilisés pour d'autres tâches, réduisant la consommation d'un centre homogène.

Le cœur des travaux d'ENTROPY adressait la problématique du placement de VM sur des serveurs physiques, sous des contraintes de consommation de ressources. Dans sa version 1.0 ce cœur permettait de minimiser le nombre de serveurs physiques utilisés. Dans sa version 2.0 ce cœur fût remanié par Fabien Hermenier, en perdant la fonctionnalité de minimisation du nombre de serveurs, mais en gagnant l'ajout de nouvelles contraintes de placement, orientées haute disponibilité.

Dans ces deux premières versions, ENTROPY a été conçu comme une boucle de reconfiguration autonome d'un centre virtualisé. Cette boucle alterne les phases d'observation du centre, d'analyse des données, de déduction d'actions et d'application de ces actions, comme présenté dans la figure 4.1. ENTROPY s'appuie sur un système d'observation de centre (initialement ganglia) pour obtenir, de manière régulière, les informations des éléments du centre. Elle utilisait aussi un système d'administration permettant d'agir sur le centre pour améliorer ce dernier (initialement des drivers spécifiques pour libvirt).

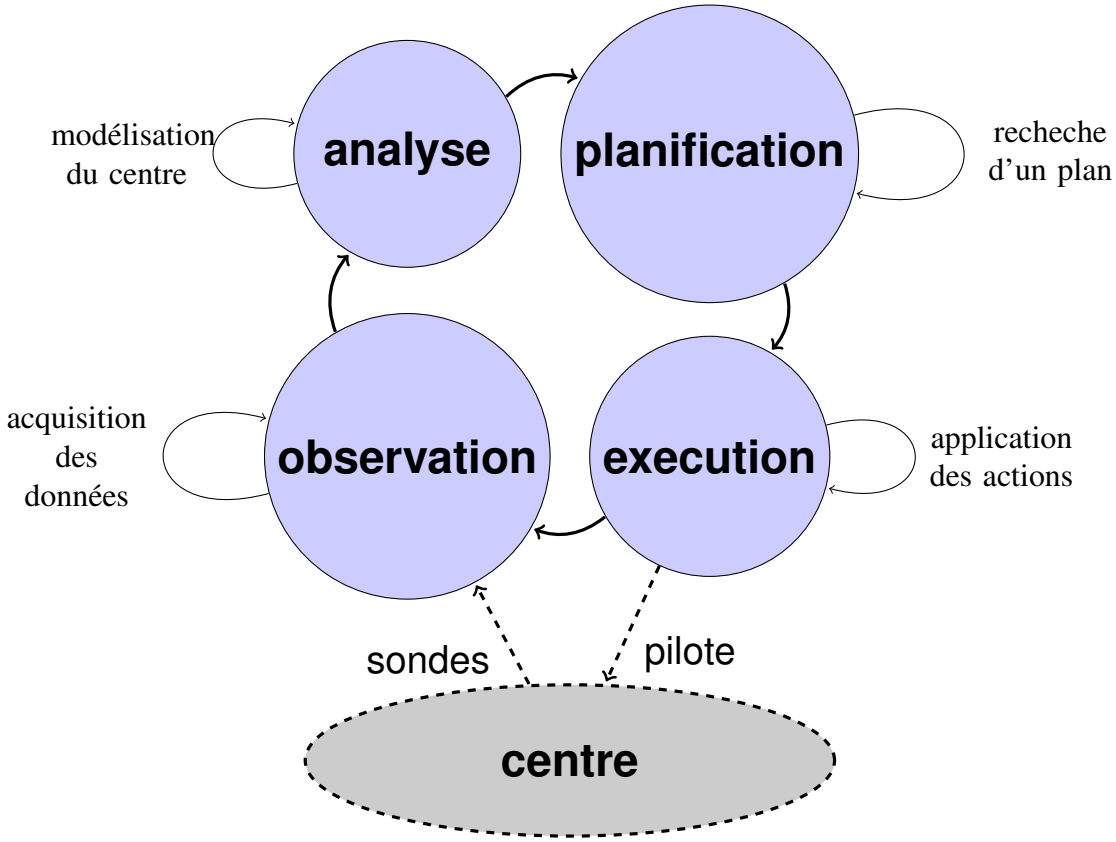


FIGURE 4.1 – Boucle de reconfiguration autonomique d'ENTROPY.

Le cœur d' ENTROPY étant axé sur la consommation des ressources, il ne permettait pas de prendre en compte les modèles de consommation électrique des serveurs. Plus généralement, il n'était pas assez flexible pour intégrer de nouvelles dimensions comme l'énergie, l'empreinte thermique, ou la gestion électrique. Or, de nos expériences passées dans le développement du système ENTROPY, il est indispensable de pouvoir intégrer au système de placement de nouvelles contraintes ou objectifs. Le choix initial d'utiliser la PPC fût essentiellement effectué pour ses propriétés de composabilité. Il est assez simple, pour un problème déjà modélisé, d'ajouter de nouvelles contraintes. Cependant, plus on ajoute de contraintes au système, plus la recherche de solution peut être difficile. Il est donc nécessaire de pouvoir activer ou désactiver un lot de contraintes en fonction des critères de placement recherchés (Thermal, HA, énergétique etc.).

C'est en développant le système d'optimisation énergétique que nous nous sommes aperçus que ce dernier devait être couplé à des contraintes diveses. Ainsi, la contrainte de type anti-affinité entre VM est nécessaire pour adresser des problématiques de haute disponibilité. Ce besoin de cohabitation avec de nouvelles contraintes fût également

important lorsque nous avons voulu intégrer, sans succès, la dimension thermique au système de placement.

Notre première contribution fût d'analyser puis de définir une architecture extensible, permettant la prise en compte de nouveaux modèles dans ENTROPY. Cette architecture nécessitait d'en modifier son cœur et donc de créer une nouvelle version d'ENTROPY. Dans les faits, différentes versions d'ENTROPY ont été développées, incompatibles entre elles bien qu'adressant pour des problèmes similaires. OPTIPLACE apporte une modularité nécessaire à la résolution de problèmes plus complexes sans modification de son cœur.

Autre point, ENTROPY V2.0 propose non seulement un choix d'actions d'administration à exécuter mais aussi leur planification. Si cette planification assure une forme de viabilité du plan de re-configuration, cet atout peut devenir un inconvénient. D'une part cette planification nécessite de prévoir la durée de chaque action d'administration possible. Or l'évaluation des durées de migration des VM dans un centre virtualisé est un problème complexe, sans solution satisfaisante [ASR+10]. Ainsi, les modèles utilisés ne peuvent qu'être approximatifs et la viabilité du plan n'est donc pas totalement assurée avec cette plannification. D'autre part, l'ajout des contraintes de planification dans un problème de re-configuration ajoute à la complexité de celui-ci. Lors du développement, le développeur de contraintes doit en comprendre les mécanismes, qu'il l'utilise ou non cette planification. Lors de la résolution d'un problème, le solveur utilisé doit considérer un compte un nombre important de contraintes, entrainant une utilisation accrue de la mémoire disponible et un temps de résolution parfois important.

Nous voulions ainsi permettre la résolution de problèmes n'intégrant pas de planification, tel un problème de répartition physique des VM. Nous voulions aussi réduire les empreintes mémoire et CPU des processus de résolution pour les problèmes de grande taille. Enfin, nous voulions rendre l'intégration de cette planification modifiable de manière extérieure, par exemple pour intégrer des algorithmes plus efficaces. Pour cela, nous devions rendre la planification dans ENTROPY optionnelle.

Il nous a fallu dans un premier temps repenser l'architecture d'OPTIPLACE. Nous avons procédé à une analyse des problèmes rencontrés lors de l'utilisation d'ENTROPY en tant que développeurs. Une fois ces problèmes cernés, nous avons déterminé les modifications que des développeurs peuvent apporter au cœur d'OPTIPLACE. Nous avons ainsi isolé les fonctionnalité minimales à conserver dans le cœur d'OPTIPLACE. Nous avons aussi déterminé les fonctionnalités à ajouter à ENTROPY pour résoudre les problèmes rencontrés.

Suite à cela, nous avons modifié ENTROPY pour lui adjoindre un système de modules, permettant non seulement de modifier le système de planification des actions mais aussi de prendre en compte des modules externes. Notre deuxième contribution fût de concevoir un module, qui d'un point de vue architectural sera dénommé une vue, de consommation énergétique. Ces travaux ont abouti à OPTIPLACE, un système modu-

laire multis-vues dédié à la gestion du placement des VM dans un centre de données virtualisé.

Le système OPTIPLACE développé dans le cadre de cette thèse se focalise sur le placement, sous contraintes et avec optimisation, des VM d'un centre virtualisé. Les composants de monitoring, de détection d'erreur ou d'action sur le centre ont fait l'objet d'une ré-implémentation dans un système plus vaste mais en dehors de cette thèse. Ce système sert de support de validation d'OPTIPLACE et est présenté ci après pour aider le lecteur à cerner les apports et limites de notre contribution.

4.1 d'Entropy 2.0 à BTRCLOUD

OPTIPLACE est intégré à la suite logicielle BTRCLOUD en tant que brique décisionnelle. Comme l'obtention des informations du centre et l'exécution des actions déduites ne sont pas du domaine décisionnel, ces fonctionnalités ne sont pas présentes dans OPTIPLACE. Le cœur d'OPTIPLACE se contente de proposer des migrations à effectuer pour améliorer un centre, en prenant en compte diverses dimensions, besoins d'optimisation et contraintes de placement internes.

4.1.1 Présentation de BTRCLOUD

BTRCLOUD est une suite logicielle d'administration de centre virtualisé issue des travaux combinés de Rémi Pottier, Frédéric Dumont, Thierry Bernard, Alexandre Garnier et Jean-Marc Menaud. Cette suite logicielle permet de gérer au travers de deux types interfaces (console et interface graphique) un centre composé de milliers de VM. Le cœur de btrCloud est défini autour de 3 composants logiciels :

- btrMonitor, en charge de la collecte des métriques et de l'architecture d'un ou plusieurs centres (Introspection)
- btrAction, en charge de la réalisation des actions logicielles sur le ou les centres (intercession)
- btrScript en charge de l'analyse des données et de la vérification des règles de placement.

Si btrScript permet de vérifier la satisfaction ou non d'une règle, ses fonctionnalités se limitent à informer l'administrateur. Il ne peut calculer un plan de re-configuration permettant de résoudre les règles de placement non respectées. Ce calcul est délégué à un module externe, appelé à la demande ou de manière automatique. Dans l'actuelle version de btrCloud, le quatrième composant en charge de ces calculs est OPTIPLACE. Dans les sections suivantes nous revenons sur les fonctionnalité et propriétés des 4 composants.

btrMonitor

Le module d'observation btrMonitor est en charge de l'observation et de l'analyse de l'infrastructure. Ce module détecte toutes les modifications dans l'organisation de l'infrastructure. L'analyse de l'infrastructure comprend donc la récupération des ressources consommées par tous les serveurs et toutes les VM, de leur cycle de vie, de l'emplacement des VM mais également de l'architecture logique de l'infrastructure (cluster, serverpool etc.).

Ce module dispose de plusieurs drivers permettant d'observer des infrastructures virtualisées sous VMWare (avec ou sans Vcenter), HyperV (avec ou sans SCVMM), CloudStack (KVM et Xen) et KVM. Ce module est également en charge de communiquer avec les différentes cartes de gestion du marché pour récupérer, entre autres, les consommations électrique et les données thermiques. Concrètement, btrMonitor interroge à intervalles réguliers, au travers des drivers de monitoring mis à disposition, les matériels spécifiés (hyperviseurs, cartes de gestion, wattmètres, etc.) afin de collecter les données de consommations accessibles sur le parc.

btrAction

Le module de reconfiguration est responsable de l'exécution des actions sur l'infrastructure en vue de modifier l'organisation des ressources. Multi-hyperviseurs, il permet de piloter des infrastructures sous VMWare (avec ou sans Vcenter), HyperV (avec ou sans SCVMM), CloudStack et KVM. Il permet l'extinction, la mise en veille et la relance des serveurs et VM du parc. Il permet aussi la migration d'une VM depuis un serveur vers un autre, la création d'un snapshot, la destruction physique d'une VM, etc..

btrScript

En charge de modéliser l'infrastructure et de vérifier la cohérence des actions d'administration, cette brique logicielle récupère les informations transmises par le module btrMonitor et envoie les actions a effectuer à btrAction. La brique btrScript est interrogeable par un administrateur via un langage dédié à l'administration d'un centre de données. L'utilisation d'un langage pour administrer une infrastructure de taille conséquente fournit une alternative intéressante aux gestionnaires graphiques de VM. En effet, un tel langage permet à l'administrateur l'utilisation de scripts et la manipulation d'ensembles d'éléments de grande taille. Le langage a été classé d'après trois sous-domaines de l'administration : la description de l'infrastructure, la sélection d'éléments au sein d'une infrastructure et la reconfiguration d'une infrastructure virtualisée.

En ce qui concerne la phase de description, une infrastructure comme un centre de données doit être perçue sous différents angles afin d'en faciliter son administration. Par exemple, une vue physique permettra de décrire de manière hiérarchique l'architecture

physique du centre de données en précisant qu'un serveur X se trouve dans une baie B elle même dans le couloir C situé dans la salle S du centre de données DC situé dans la ville V. Une vue électrique permettra de décrire que la prise P est connecté au disjoncteur D du tableau basse tension T alimenté par l'arrivée A. Ces vues sont génériques et paramétrables en fonction du contexte d'utilisation de btrCloud. Une fois les vues décrites, il est alors aisé pour un administrateur de manipuler ces abstractions et de déterminer, par exemple, quelle est la consommation instantanée des serveurs connectés sur le disjoncteur D. Nous avons réutilisé cette notion de vue dans la conception de l'outil OPTIPLACE.

En ce qui concerne le langage de manipulation et d'action, nous proposons au lecteur de consulter la thèse de Rémy Pottier [Pot12].

OptiPlace

La brique logicielle OPTIPLACE interagit avec la brique logicielle btrScript. Quand ce dernier détecte une règle de placement non satisfaite, il fait appel automatiquement ou semi-automatiquement au système de résolution de contraintes OPTIPLACE. Il est possible de faire à appel à OPTIPLACE soit à la demande de l'administrateur, soit à une fréquence régulière, soit enfin en réaction à des événements particuliers. Le rôle d' OPTIPLACE est à partir d'une configuration donnée (l'état courant du système) et de règles de placement, de générer un plan de reconfiguration (une suite d'action à exécuter) permettant d'aboutir à un placement respectant les règles de placement et minimisant/maximisant la fonction de coût.

Comme décrit, la notion de vue présente dans btrScript est également disponible dans OPTIPLACE. Ces vues permettent d'ajouter au problème de placement principal, des notions, règles et objectifs propres au problème modélisé par la vue. La description de cette architecture est détaillée dans la section suivante.

4.2 OPTIPLACE

L'objectif premier d'OPTIPLACE est la résolution d'un problème de placement de VM sur des serveurs physiques. Cet outil initial peut être étendu via la notion de *vue*.

Initialement le système est construit sur un cœur minimal, qui correspond au problème générique de placement à résoudre. Dans ce problème, OPTIPLACE considère une configuration virtuelle d'un centre, c'est à dire les caractéristiques des serveurs et VM du centre, ainsi que l'hébergement des VM sur les serveurs. La résolution d'un problème de placement permet d'obtenir, si nécessaire, une nouvelle configuration de ce centre.

Le cœur d'OPTIPLACE ne prend initialement pas en compte de contraintes, et se contente de placer les VM sur un ou plusieurs serveurs. Ce cœur peut prendre en compte

un premier niveau de contraintes : les consommations des ressources virtuelles dans le centre. En effet les VM utilisent (par exemple le CPU) ou réservent (par exemple la mémoire) les ressources physiques des serveurs. OPTIPLACE permet de spécifier des capacités et consommations de ressources dans le centre, et s'assure ensuite que les ressources consommées par les VM hébergées par un serveur ne dépassent pas les capacités de ce serveur. Ce problème d'allocation sous contraintes de ressources correspond à un problème de *binpacking*. Ce binpacking est mono-dimensionnel quand on ne considère qu'une seule ressource, multi-dimensionnel quand on en considère plusieurs. Le système étant modulaire, le cœur initial peut considérer de 0 à autant que l'on veut de ressources. Par défaut, les ressources CPU et mémoire sont ajoutées au problème.

Le cœur d'OPTIPLACE résout un problème de placement en ne considérant que les serveurs, les VM, et la configuration initiale des VM sur le serveurs. Comme nous venons de voir, le placement des VM dans un centre est contraint par les capacités et utilisations de ressources dans celui-ci. Le problème initial peut être enrichi grâce à l'ajout de vues. Une vue apporte au problème de placement des données des modèles, règles et objectifs supplémentaires à prendre en compte. Les vues sont activables et désactivables à volonté, étant un élément externe au cœur. Ainsi, selon l'objectif qu'il se fixe, par exemple minimiser la consommation électrique ou améliorer la disponibilité du système, et selon les spécificités du centre sur lequel il travaille, l'administrateur peut n'activer que les vues qui l'intéressent.

Dans le cadre de nos travaux, cette flexibilité n'est pas seulement nécessaire pour l'adaptabilité, la généricité ou la maintenabilité du code, mais aussi pour des problèmes de performances. Rajouter des données et des contraintes dans le solveur complexifie le problème à résoudre et augmente donc le temps de réponse. Si ces données et contraintes ne sont pas utiles pour l'objectif à atteindre, il est important qu'elles ne soient pas présentes dans le solveur. Ainsi, les règles et objectifs non minimaux sont déportés dans les vues afin d'améliorer le temps de réponse lorsqu'ils ne sont pas nécessaires.

Outre une vue intégrant les ressources CPU et mémoire, la première vue que nous avons développée a été la vue HD. Initialement les règles développées dans le cadre des travaux de Fabien Hermenier, pour permettre au système de résoudre un placement de VM sous des contraintes de haute disponibilité, étaient entremêlées dans le coeur d'ENTROPY. Nous avons externalisé ces règles dans un module HD, qui propose des règles de placement des VM les unes par rapport aux autres.

La première contribution est présente sous la forme d'un travail d'architecture du système pour dissocier le cœur minimal d'OPTIPLACE du module HD. Ce mode d'utilisation d'OPTIPLACE a demandé des modifications importantes dans l'architecture d'ENTROPY. Un système de *vues* permet aux développeurs de spécifier leurs propres règles modifiant le problème de placement initial. Ils peuvent ainsi y ajouter des informations sur l'état des éléments du centre et des ressources, ou proposer des règles et objectifs à utiliser par l'administrateur. En particulier, l'architecture modulaire permet

de développer des dictionnaires de contraintes ou des modèles propres à des catégories de problèmes et de les réutiliser dans d'autres développements sans toucher au cœur d'OPTIPLACE. Nous avons également rajouté à la vue HD initiale quelques nouvelles règles.

Un exemple de vue est la planification temporelle, telle que présente dans ENTROPY, ou encore la vue énergétique qui a guidé notre développement. Cette dernière incorpore les consommations électriques des serveurs commes fonctions de l'utilisation des ressources de ce serveur. La deuxième contribution fût ainsi de créer une vue énergétique, intégrant des modèles, règles, coûts et stratégies pour considérer la consommation électrique des serveurs du centre.

Le principe générique est détaillé dans la section suivante. La vue énergétique développée dans cette thèse est décrite dans la section 4.3.

4.2.1 Programmation modulaire : les vues

Les vues sont, d'un point de vue administrateur, de nouvelles règles que l'on veut ajouter au cœur minimal. Ces règles peuvent porter sur les VM, sur les serveurs comme celles déjà présentes dans ENTROPY, ou éventuellement sur des entités spécifiques à une vue. Elles doivent être respectées dans la configuration fournie par OPTIPLACE.

En plus des règles, une vue peut apporter une notion de coût de solution à réduire lors de la recherche d'une solution à un problème. Ce coût est une fonction mathématique à minimiser lors de cette recherche, et modélisée à partir des variables du problème. Par exemple, dans ENTROPY V2.0 l'objectif était de minimiser le nombre de migration dans le plan de reconfiguration.

Ces vues étendent le cœur minimal d'OPTIPLACE, en ajoutant de nouvelles données au problème initial de placement. Typiquement la vue énergétique ajoute, en plus des spécifications des ressources du centre, des modèles de consommation électrique de ces serveurs en fonction de la consommation des ressources des VM qu'ils hébergent. Elle est utilisée comme une extension de la configuration virtuelle, ajoutant ses variables, contraintes, et coûts au problème de placement. Ainsi une vue s'appuie sur les quatre piliers de la PPC : les variables du problèmes, les plages de valeur de ces variables, les contraintes entre ces variables, et les potentielles fonctions de coût à minimiser.

La première vue HD a principalement demandé un travail de ré-architecture. Cette vue n'étend pas la configuration initiale, car elle n'apporte pas de données supplémentaires au modèle initial. Elle donne accès à des règles de placement : spread, group, ban, fence, issues des travaux sur ENTROPY V2.0. Dans le cas où ou le système recherche un placement valide pour toutes les VM, il n'y a pas de fonction de coût associé. Cependant dans le cas où le système cherche à réduire le nombre de migrations, le vue HD propose (importé d'ENTROPY V2.0) une fonction de coût qui est le nombre total de migrations à effectuer pour passer de la configuration initiale à la solution. L'intérêt de cette vue

HA consiste plus en son apport architectural que dans ses fonctionnalités, qui sont les mêmes que celles présentes dans ENTROPY V2.0.

Concernant la vue énergétique, un travail de recherche et d'évaluation plus important à été nécessaire, décrit dans la section correspondante 4.3.

Utilisation par l'administrateur

Un administrateur utilise OPTIPLACE pour maintenir son centre dans un état satisfaisant. Cette notion de satisfaction est intrinsèque au centre considéré, en effet chaque centre possède ses règles spécifiques. Le problème du placement des VM dans un centre doit donc être paramétré par l'administrateur pour prendre en compte ces spécificités.

La configuration virtuelle du centre, c'est-à-dire les listes de VM, serveurs et les allocations des VM sur les serveurs, peut être généralement obtenue par le système de gestion du centre. Cependant, les demandes spécifiques de l'administrateur, telles l'arrêt d'un serveur ou la réduction des points chauds du centre, ne sont pas connues par de tels systèmes de gestion. De même, les utilisateurs du centre peuvent spécifier des demandes à prendre en compte par l'administrateur, telles la création d'une nouvelle VM ou l'augmentation des capacité en mémoire d'une VM existante.

Le système de vues permet, une fois paramétré, de prendre en compte ces demandes spécifiques sous la forme de règles à injecter dans un problème de placement. OPTIPLACE permet à l'administrateur d'un centre d'ajouter des règles de fonctionnement et de préciser un objectif à atteindre, sous la forme d'une fonction de coût. Pour cela, les vues doivent tout d'abord obtenir les données propres à leur domaine. Une fois correctement configurées, les vues sont utilisées pour chaque problème de placement dans un centre. À chaque fois qu'OPTIPLACE doit résoudre tel problème, les vues activées par l'administrateur injectent leurs règles dans le cœur du problème. Par exemple, si la vue énergétique est activée, elle peut ajouter ses modèles de consommation des serveurs dans le problème. En particulier, ce sera le cas si l'administrateur veut réduire la consommation du centre ou de certains serveurs.

Le paramétrage des vues, en particulier l'obtention des données propres à leur domaine, est une opération à la charge de l'administrateur.

Intégration des vues à la résolution d'un problème

Le processus de résolution d'OPTIPLACE, présenté dans la figure 4.2, est le suivant :

Tout d'abord, OPTIPLACE nécessite de connaître la configuration virtuelle du centre, tel que décrite plus haut. Il peut ainsi modéliser en interne cette configuration, c'est-à-dire le cœur du problème de placement des VM sur les serveurs. Cette information est typiquement obtenue par BTRCLOUD après traduction des indications des gestionnaires dans le modèle d'OPTIPLACE, via son API.

Une fois le cœur du problème créé, celui-ci est enrichi des spécifications de ressources. Ces spécifications sont fournies par les vues et décrivent les capacités des serveurs et les consommations des VM pour chaque ressource. Des fonctionnalités dédiées réduisent la charge de développement inhérente à l'ajout d'une ressource.

Les vues paramétrées par l'administrateur sont ensuite associées au problème. Celles-ci accèdent ainsi aux données du problèmes et ajoutent leurs variables et contraintes dans ce dernier. Par exemple, la vue énergétique utilise les variables d'utilisation des ressources des serveurs pour créer et contraindre des variables de consommation électrique.

Une fois ces vues associées, leurs règles sont injectées dans le problème. Ces règles sont appliquées aux variables du problème, que ce soit celles crées par le cœur ou par les vues, en ajoutant des contraintes dans le cœur du problème. Par exemple le bannissement d'un serveur, demandé par l'administrateur pour des raisons de maintenance, contraint le nombre de VM hébergées sur un serveur à être zéro.

Enfin, une fois les règles injectées dans le problème, l'objectif de recherche (c'est-à-dire le coût à réduire) est déterminé et utilisé dans la résolution du problème. Cette résolution consiste en la recherche d'un état du centre qui respecte les règles des vues, tout en minimisant le coût spécifié. Quand le solveur à déduit du problème une solution, les vues utilisées en déduisent les résultats qui leurs sont propres. Par exemple, la valeur de consommation totale des serveurs du centre est déduite par la vue énergétique, la seule à avoir connaissance du sens de cette valeur.

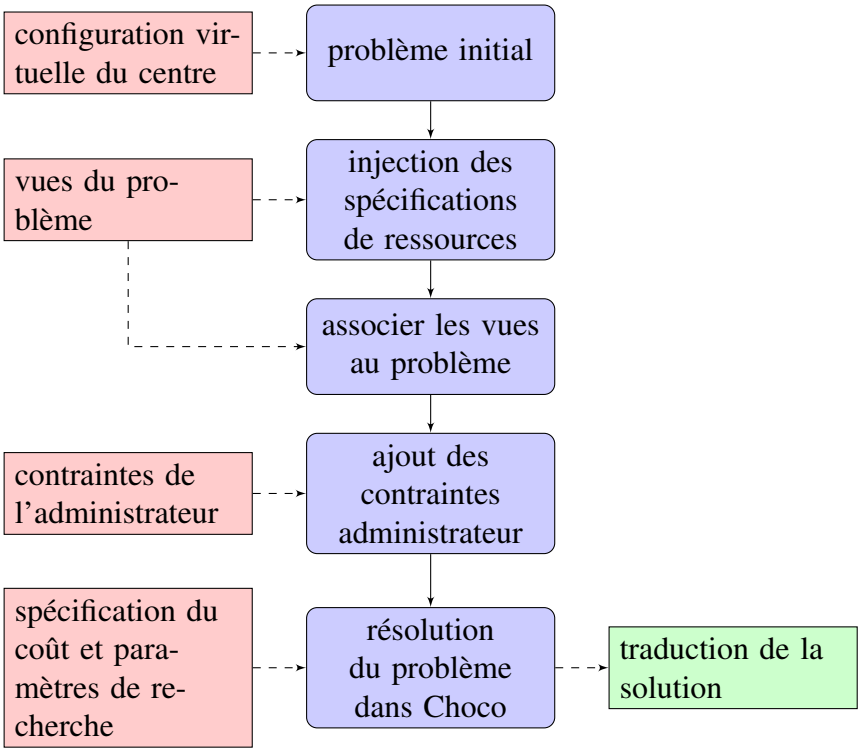


FIGURE 4.2 – Processus d'utilisation d'OPTIPLACE.

Ainsi, les vues exposent leurs variables et contraintes, qu'elles les utilisent directement ou que celles-ci soient utilisées par d'autres vues, tout en proposant à l'adminsitrateur des méthodes pour enrichir le problème de placement.

L'avantage de cette modularité est que les paramètres du problème et les règles de différentes vues peuvent être incorporées simultanément dans un problème. Il est donc possible de développer une nouvelle vue sans aucune connaissance des vues développées précédemment ou même activées dans le système, et par la suite de changer une vue pour une version plus à jour sans modifier les autres vues ni les paramètres d'exécution du problème.

Cette fonctionnalité apporte la flexibilité nécessaire à un outil de placement intelligent et efficace.

Intégration des vue dans OPTIPLACE

Le cœur d'OPTIPLACE permet de définir la configuration virtuelle d'un centre, les paramètres de résolution d'un problème, et présente les fonctions d'extension par des vues.

Les vue peuvent s'appuyer sur d'autres vues pour s'intégrer dans un problème. Ainsi, la vue thermique, qui modélise le coût de la climatisation, nécessite des modèles de consommation des serveurs apportés par la vue énergétique . Lorsqu'un développeur

améliore la vue énergétique, ces améliorations sont apportées à la vue thermique sans avoir besoin de modifier cette dernière. Pour cela, chaque vue doit présenter de manière explicite les variables qu'elle ajoute au problème, ainsi que les contraintes qu'elle peut générer.

En suivant ces besoin, nous avons développé OPTIPLACE pour que les vues accèdent dans le cœur d'un problème à des fonctions communes de création de variables et de contraintes. Une fois paramétrée, une vue est alors indépendante d'un quelconque problème de placement, bien qu'elle soit basée sur une première configuration virtuelle. C'est pourquoi, lors de la résolution d'un problème, chaque vue activée est associée au cœur de ce problème. Une fois associée, une vue a accès aux variables du cœur du problème, ainsi qu'aux fonctions précédemment décrites. Elle peut alors utiliser ces dernières pour intégrer ses règles, sous formes de contraintes, au problème.

En plus des contraintes à satisfaire, les vues peuvent spécifier une fonction de coût à minimiser lors de la résolution d'un problème. Une telle fonction de coût est intégrée dans un problème de placement par la création d'une variable correspondante dans le problème.

Ces fonctionnalité d'injection de règles, de spécification des ressources et de sélection d'un coût dessinent les traits d'une architecture réflexive. Cette capacité des vues à changer le modèle interne ne se limite pas à l'ajout de variables et contraintes, mais aussi à l'accès à des paramètres plus poussés de recherche de solution.

4.2.2 Vers un système réflexif

Si la spécification d'une fonction de coût associée à la recherche des solutions d'un problème permet d'orienter cette recherche, la PPC propose de plus l'accès à des fonctionnalités liées à son principe de fonctionnement, c'est-à-dire le parcours d'un arbre de recherche.

En particulier, la résolution d'un problème générique ne peut être résolue efficacement avec une stratégie générique. Il faut donc adapter les mécanismes de résolution d'un problème à ce dernier, c'est-à-dire permettre au développeur de vues de modifier depuis celles-ci le fonctionnement du solveur.

Réduction de la fonction de coût

Une fonction de coût spécifie un objectif à atteindre lors de la résolution d'un problème, sa sémantique se place donc non pas dans le cadre du centre considéré mais uniquement dans le cadre d'une recherche de solution. En cela la sélection de cette fonction est une forme de réflexion. Nous poussons plus loin cette fonctionnalité en permettant de paramétrer l'intégration de cette fonction de coût au problème.

En effet lorsque les modèles considérés sont imprécis, deux solutions dont la différence de coût est de l'ordre de cette précision peuvent être considérées comme équi-

valentes. Par exemple, si la fonction de coût est précise à 20% près, alors une solution dont le coût est 100 et une solution de coût 90 peuvent être considérées équivalentes dans l'arbre de recherche. Quand une solution est trouvée par le solveur, la recherche des solutions équivalentes à une précision près n'est plus intéressante. Il est alors possible d'élaguer l'arbre de recherche en éliminant au plus tôt les solutions dont le coût est équivalent.

Notre approche permet de spécifier une fonction de réduction du coût. Dans notre exemple précédent, dès que l'exploration d'un arbre obtient une solution dont le coût vaut 200, cette fonction de réduction est alors appelée. Elle indique au solveur que celui-ci doit désormais chercher des solutions dont le coût est inférieur à $\frac{100-20}{100} \times 200 = 160$. Cette fonction de réduction réduit ici le coût d'une solution trouvée de 200 à 160.

Cette réduction considère dans notre exemple une précision relative et consiste en un simple multiplicateur de coût. Un développeur peut cependant spécifier des réductions de coût plus complexes, par exemple une homothétie centrée en $\frac{\beta}{1-\alpha}$ et de coefficient α de formule $r(c) = \alpha \times c + \beta$ ou un déplacement absolu de vecteur δ de formule $r(c) = c + \delta$. Ces trois représentations sont intégrées au cœur d'OPTIPLACE mais celui-ci permet à un développeur d'intégrer ses propres formules.

En pratique, l'utilisation de cette fonctionnalité réduit l'espace de recherche, mais seulement quand la la formulation du coût le permet. En effet, il n'est pas toujours possible d''éliminer des branches de l'arbre des solution à partir de la fonction de coût. Cette possibilité dépend principalement de l'intégration de la fonction de coût au problème. Ainsi la réduction d'une fonction de coût très complexe n'a qu'un impact très réduit sur la recherche des solutions par le solveur.

Cette approche ne permet plus d'obtenir la meilleure solution, mais une solution satisfaisante, qui dans certains contextes tels la réduction énergétique est suffisante.

Paramétrage de l'injection des règles

Le cœur d'OPTIPLACE ne permet de déclarer qu'un seul type de contraintes : les contraintes de satisfaction des ressources, générées automatiquement à partir des spécifications d'utilisation des ressources serveur par les VM. Cette contrainte de packing des ressources était au cœur d'ENTROPY. Bien que formellement très simple, elle peut être traduite de différentes manières dans un problème de PPC. Plus particulièrement, le comportement associé à la mise à jour des variables de la contrainte, aussi appelé propagation de la contrainte, peut être adapté à différentes classes de problèmes.

La méthode d'injection de ce type de contrainte peut être spécifiée dans le cœur. Ainsi l'administrateur peut utiliser sur un problème à contraintes spécifiques une implémentation de packing à faible recherche d'optimisation. De la même manière il peut utiliser un algorithme plus intelligent pour un autre problème, qui analysera le système sous tous ses angles.

Ce système de packing peut être spécifié par l'administrateur du centre, ou bien par

le développeur de vues. En effet, les vues peuvent communiquer entre elles, puisqu'une vue peut être construite en s'appuyant sur les fonctionnalités d'une autre vue. Il est ainsi possible de fournir des vues fonctionnelles pures, c'est à dire ne fournissant que des fonctions de manipulation du problème sans altérer celui-ci par des données ou règles propres.

Ainsi des fonctionnalités de contraintes spécifiques, telles du calcul matriciel, peuvent être ajoutées à OPTIPLACE sans changer son noyau. Ces vues peuvent être échangées pour d'autres vues proposant les même fonctionnalité sans rompre la cohérence du problème. Ces fonctionnalités peuvent être utilisées pour la génération de contraintes liées à une vue ou la création de stratégies de recherche associées à un problème.

Stratégies de recherche

Une adaptation usuelle d'un solveur de PPC au problème qu'il résout passe par la spécification d'une stratégie de recherche, ou heuristique. Dans un problème de PPC, une heuristique permet de choisir quelle branche traiter en priorité dans l'exploration de l'arbre de recherche. Elle est issue de l'expérience de résolution d'un problème et permet d'ajouter à ce problème des connaissances qui ne peuvent être exprimées par la PPC.

Le solveur de PPC Choco, utilisé par le cœur d'OPTIPLACE, permet de définir de telles stratégies. Nous utilisons ses modèles pour permettre la création, à partir des éléments des vues, des heuristiques à utiliser lors de l'exploration de l'arbre des solutions. Elles peuvent être spécifiées par un développeur, ou indirectement par la fonction de coût associée au problème.

Le paramétrage direct par le développeur permet de comparer facilement les performances de stratégies proches, en résolvant deux fois le même problème avec ces deux stratégies successives.

Performances du solveur

L'effet des paramètres de résolution d'un problème peut être déduit des performances du solveur lors de la résolution d'un problème. Celles-ci sont mises à la disposition des développeurs après la résolution d'un problème, permettant de comparer facilement les paramétrages. Sont ainsi présentés les durées de création et résolution du problème, de même que les nombres de contraintes et variables crées par chaque vue ajoutée au problème. Ces valeurs permettent d'évaluer les compromis induits par les stratégies de recherche, par exemple celles que nous avons développées pour la vue énergétique. Elles permettent aussi de d'évaluer l'intégration commune de plusieurs vues.

S'il est vrai que les vues peuvent être mélangées dans OPTIPLACE, la PPC n'autorisant qu'une fonction de coût pour un problème, il n'est possible de minimiser qu'une seule fonction de coût. Ceci peut poser problème pour prendre en compte plusieurs coûts

à réduire dans un même problème. Dans ce cas, il y a deux solution : ou bien créer une fonction de coût qui soit un amalgame des autres fonctions de coût, ou bien développer une vue définissant sa propre fonction de coût composite. Dans le deuxième cas, si la composabilité de la PPC assure un fonctionnement correct, il n'en est pas de même pour les performances du solveur.

4.3 La vue énergétique

Comme précisé, la consommation des centres est un problème important, puisqu'une estimation de 2011 indique que celle-ci atteindrait 1.5% de la consommation électrique mondiale [Koo11]. La grande quantité de serveurs en fonctionnement, les besoins croissants en services informatiques et en particulier l'avènement du cloud computing, les besoins de redondance pour des raisons de sécurité sont autant de facteurs de croissance, que le gain d'efficacité électrique des technologies récentes ne parvient pas à équilibrer.

De plus, à la consommation des serveurs du centre s'ajoute la consommation de l'équipement électrique annexe qui contribue de manière non négligeable à la consommation d'un centre. L'objectif de réduction des coûts énergétiques est maintenant un problème pris en compte par les architectes lors de la création d'un centre. Cependant l'évolution d'un centre, pour cause de changement de matériel ou pour accroître ses capacité de service, réduit l'efficacité énergétique de celui-ci. En particulier, le matériel de climatisation d'un centre, chargé de climatiser l'atmosphère et de refroidir les serveurs, peut être responsable de jusqu'à 50% de la consommation d'un serveur.

L'utilisation de certaines technologies permet d'améliorer ce problème de consommation. D'une part, la virtualisation des services offre des capacités de mutualisation des ressources, réduisant les nombre de serveurs allumés pour exécuter un jeu de services. D'autre part, les systèmes de refroidissement alternatifs utilisant l'air extérieur ou de l'eau promettent une réduction importante du coût de la climatisation. Si ces technologies d'architecture du centre ou d'exécution des services permettent de réduire l'impact environnemental d'un centre, une gestion intelligente de celui-ci est nécessaire pour utiliser ces technologies à bon escient.

ENTROPY prenait en compte le problème de consommation en permettant à l'administrateur d'éteindre un maximum de serveurs. Nous proposons de prendre en compte les modèles de serveurs installés dans le centre pour permettre l'exécution des services sur les serveurs les plus efficaces d'un point de vue énergétique. Ces travaux permettent de plus de s'assurer de la modularité de notre architecture.

Intuitivement, la consommation (électrique) d'un serveur est croissante avec l'utilisation de ses ressources. Plus les services exécutés demandent de calculs à effectuer, d'écritures sur le disque, de paquets réseau à envoyer, et plus la consommation du serveur est élevée. Cependant, les différentes technologies utilisées dans les serveurs, et en particulier les usages pour lesquels ils ont été prévus, induisent des différences d'effica-

cité énergétique parmi les serveurs d'un centre hétérogène.

La vue énergétique lie les consommations électriques des serveurs à l'activité des VM qu'ils exécutent. Nous déterminons la consommation de tout ou partie des serveurs d'un centre en fonction de la configuration virtuelle choisie, et utilisons ces valeurs de consommation lors de la recherche d'une amélioration du centre. Dans cette vue nous ignorons le matériel qui n'est pas utilisé pour la virtualisation, nous ne considérons que les serveurs et les VM que ceux ci hébergent, décrivant la fonction de consommation du centre :

$$cons(centre\ c) = \sum_{serveur\ s \in c} cons(s)$$

De plus, nous considérons la consommation des serveurs sur un régime stable, *eg.* sur une minute. Dans ce cas, nous considérons la puissance moyenne consommée par un serveur sur une période, *eg.* la différence d'énergie consommée par le serveur sur une période divisée par la durée de cette période.

4.3.1 Les données du problème

Cette vue ajoute au problème de placement initial des données qui ont trait à la consommation électrique des serveurs. Pour cela elle dispose de modèles de consommation des serveur, qu'elle injecte dans le problème de placement. Cette injection contraint une variable représentant la consommation du serveur, selon le modèle qui lui est associé.

Elle propose ensuite des fonctions de coût à réduire, dans le but de minimiser, équilibrer ou restreindre la consommation des serveurs dans le centre. Ces fonctions sont activées par d'autres vues, selon le paramétrage de l'administrateur. Minimiser la consommation électrique correspond au problème de réduction du coût de fonctionnement des serveurs, tandis que l'équilibrage des charges est une réponse à la réduction des charges thermiques dans un centre.

Lorsque le centre n'a pas la capacité physique pour dissiper la consommation électrique maximale des serveurs, que ce soit pour des raisons d'évolution du parc de machines ou de pannes du système de climatisation, l'administrateur peut limiter la consommation d'un groupe de serveurs. Dans ce cas l'administrateur spécifie une consommation maximale à ne pas dépasser dans un groupe de serveurs lors de la résolution d'un problème de placement. La solution proposée se base sur le placement intelligent des VM, nous n'agissons pas sur les mécanismes de contrôle au niveau serveur tel le système de gestion du DVFS.

Le centre peut aussi être sujet à des problèmes de "points chauds" dus à des défauts de ventilation. Dans ce cas l'administrateur peut borner la consommation de certains serveurs au regard de celles d'autres serveurs. Ainsi, les serveurs les moins ventilés auront nécessairement une charge électrique moins importante que celle des serveurs les plus ventilés.

Les consommations des serveurs sont liées non seulement aux activités de leurs VM, mais aussi aux modèles physiques des serveurs installés. De plus, certains paramétrages logiciels permettent de réduire de manière non négligeable la consommation d'un serveur à faible demande de travail. Par exemple, le DVFS réduit la consommation d'un CPU lorsque celui-ci est utilisé à faible charge. Dans les centres où se côtoient différents modèles de serveurs, il n'est donc pas réaliste de supposer que tous les serveurs aient le même modèle de consommation. Au contraire, il devient indispensable de modéliser les consommations de chaque serveur afin de les prendre en compte dans le système de placement.

Cette vue ajoute de nouvelles variables au problème initial, en associant à chaque serveur une valeur de consommation. Cette valeur est contrainte par le modèle de consommation du serveur, et des activités ressources fournies par le cœur d'OPTIPLACE. La modélisation des consommations des serveurs est un problème abordé sous de multiples angles dans la littérature. Nous ne cherchons pas dans cette thèse à résoudre ce problème, puisque nous nous concentrons sur la prise en compte de ces modèles dans la vue énergétique.

Dans certains centres, l'administrateur a la possibilité d'éteindre les serveurs inutilisés afin de réduire la consommation du centre. Si cette extinction permet de réduire grandement la consommation du centre, elle peut compromettre les performances de celui ci. Ainsi si un serveur devient surchargé, le temps de migration d'une VM de ce serveur vers un serveur éteint nécessite le démarrage du serveur de destination. Dans certains cas, cette fonctionnalité est tout simplement désactivée car non prise en compte par le SGIV . Cette possibilité est prise en compte dans la vue consommation et, si activée, rend la consommation d'un serveur égale à 0 quand celui-ci n'héberge aucune VM. Elle est dissociée des modèles de serveurs car elle dépend uniquement de la capacité à éteindre des serveurs dans le centre.

La littérature présente de nombreuses classes de modèles de consommation, linéaire, par paliers, quadratique selon les serveurs considérés et les activités du centre. La vue énergétique d'OPTIPLACE doit connaître, pour chaque serveur, son modèle de consommation et fournit pour cela une interface générique de modèle à respecter. Cette interface générique permet l'injection d'un modèle de consommation la respectant dans un problème de placement à la demande. L'administrateur du centre doit donc spécifier et paramétrer le modèle de chaque serveur, que ce soit de manière statique ou par un apprentissage dynamique. Nous nous concentrons sur l'implémentation des modèles car le paramétrage des modèles rejoint le problème de modélisation des serveurs et sort du cadre de cette thèse.

4.3.2 Modèles de consommation des serveurs

La consommation d'un serveur, sous la forme d'une puissance, est souvent mise en relation avec la charge de celui-ci. Lorsque le serveur n'est pas virtualisé et que les ap-

plications que celui ci exécute sont connues, le terme de charge du serveur a un sens relatif aux applications présentes. Par exemple, la charge d'un serveur web peut être définie comme le nombre de requêtes que celui ci reçoit par secondes, celle d'un encodeur de flux comme le nombre d'octets à traiter par secondes ou la charge d'un serveur HPC comme le nombre de cœurs CPU monopolisés par son exécution. Cependant les serveurs virtualisés n'ont pas connaissance des applications qui sont présentes dans leurs VM. Dans ce cas la notion de charge d'un serveur se limite à considérer les accès aux ressources de celui-ci, et plus particulièrement les accès ressources effectués par les VM qu'il héberge. La vue énergétique permet de spécifier des modèles de relation entre la consommation électrique d'un serveur et la configuration virtuelle du centre.

Si la littérature ne permet pas de dégager un modèle général de consommation des serveurs, elle apporte cependant différentes classes de consommation de serveurs. Chaque classe demande un développement spécifique pour être incorporée dans OPTIPLACE, et influence les performances de résolution des problèmes de placement. En effet, des modèles simples permettent au solveur de déterminer rapidement la consommation d'un serveur, tandis que les modèles plus compliqués nécessitent une connaissance approfondie des VM présentes sur le serveur pour déterminer sa consommation.

Nous ne détaillerons pas le modèle de consommation constante, qui bien que très simple à réaliser(la consommation du serveur est fixée) n'a que peu de sens aujourd'hui, tous les serveurs ayant une forme de réduction de la consommation à basse charge. Nous présentons quelques modèles que nous avons développés pour l'occasion, qui peuvent être enrichis ou modifiés par les développeurs pour correspondre aux observations du centre.

Modèle linéaire en CPU

Le modèle le plus fréquemment cité est le modèle linéaire en CPU. Selon ce modèle, la puissance consommée par le serveur est une fonction linéaire de la charge CPU de ce dernier.

$$P_{lin,CPU}(serveur\, s) = \beta + CPU(s) \times \alpha$$

Dans ce cas, le coefficient de croissance α et la charge à vide β, de valeurs positives, sont propres propres à chaque serveur.

Nous normalisons la charge CPU du serveur considéré, pour qu'elle atteigne 1 au maximum, quelque soit l'unité obtenue par le SGIV (MHz, % de temps utilisé, nombre de cœur utilisés, MIPS, etc.). Pour cela nous divisons la charge observée par la charge maximale (observée ou connue) et multiplions le coefficient de croissance par cette charge maximale. De ce fait la consommation du serveur varie entre β et $\alpha + \beta$ de manière linéaire, nous posons P_{min} et P_{max} les consommations respectivement minimales et maximales du serveur. Dans ce modèle et avec une charge normalisée, ces valeurs valent respectivement $P_{min} = \beta$ et $P_{max} = \alpha + \beta$.

Ce modèle a été intégré dans OPTIPLACE en utilisant la formule suivante :

$$P_{lin,CPU}(serveur\ s) = \frac{CPU(s) \times (P_{max} - P_{min})}{CPU_{max}} + P_{min}$$

En effet, les systèmes d'observation présents dans BTRPLACE permettent d'observer les activité CPU des serveurs et leurs consommations électriques. Un tel modèle de consommation peut donc être

Consommation des VM sous modèle linéaire

Dans le cas d'une consommation linéaire avec l'activité du serveur, il est possible d'associer à une VM sa consommation apportée au serveur. Cette valeur représente l'augmentation de consommation induite par l'hébergement de cette VM sur ce serveur, et ne dépend que des activités de cette VM. Ainsi dans le modèle linéaire en CPU nous avons :

$$P_{\delta}(vm, s) = CPU(vm) \times \frac{(P_{max}(s) - P_{min}(s))}{CPU_{max}(s)}$$

Cette valeur $P_{\delta}(vm, s)$ représente le coût énergétique de l'affectation d'une VM à un serveur. La consommation du serveur peut être calculée en intégrant cette notion. Nous définissons $h(s, vm)$ la variable booléenne d'affectation de la VM vm au serveur s, c'est-à-dire $h(s, vm) = 1$ ssi vm est hébergée par s, 0 sinon. La consommation d'un serveur est alors la somme, pour chaque VM que celui-ci héberge, des consommations induites par ces VM sur ce serveur, à laquelle nous ajoutons la consommation minimale du serveur :

$$P(s) = P_{min}(s) + \sum_{vm} h(s, vm) \times P_{\delta}(vm, s)$$

Quand tous les serveurs du centre c ont une consommation linéaire il est possible de redéfinir la consommation du centre comme somme des consommations des VM ajoutée à la somme des consommation minimale des serveur.

$$P(c) = \sum_{s} P_{min}(s) + \sum_{s,vm} h(s, vm) \times P_{\delta}(vm, s)$$

On définit la consommation induite par vm dans le centre comme étant les éléments de cette somme relatifs à vm :

$$P_{\delta}(vm) = \sum_{s} h(s, vm) \times P_{\delta}(vm, s)$$

Nous obtenons une définition de la consommation qui prend en compte les consommations minimales des serveurs et les consommations induites par les VM :

$$P(c) = \sum_{s} P_{min}(s) + \sum_{vm} P_{\delta}(vm)$$

Étant donné que la consommation minimale des serveurs est une valeur constante, ce changement de définition permet de réduire les éléments variables du problème.

Cette reformulation est intéressante pour déterminer la consommation minimale du centre. Elle permet de déduire au plus tôt une borne inférieure de la consommation des VM, et donc du centre. Elle demande cependant l'utilisation de contraintes supplémentaires dans le problèmes de placement, qui sont bien plus coûteuses en terme de performances que les contraintes d'évaluation du modèle de consommation linéaire. Aussi cette formulation est réservé à une recherche exhaustive des solutions du problème de placement, ou à un problème fortement contraint sur les consommations des serveurs.

Ce modèle linéaire souffre d'un manque d'adaptabilité, il est en effet peu probable qu'un serveur n'utilise qu'une seule ressource durant son utilisation. Il permet cependant de réutiliser les résultats scientifiques sur la consommation électrique des différents composants d'un serveur, et d'obtenir une base de comparaison avec d'autres modèles plus compliqués.

Consommation linéaire multi-ressources

Le modèle linéaire en CPU est un des principaux modèles mis en avant dans la littérature. Cependant, la ressource CPU telle que spécifiée dans OPTIPLACE peut avoir différents sens, par exemple, l'administrateur d'un centre peut réserver pour certaines VM un certain nombre de cœurs CPU. Dans ce cas, la notion de ressource CPU réservée dans OPTIPLACE est dé-corrélée de la notion d'activité CPU visible depuis le SGIV.

De même, des serveurs orientés encodage de flux ou résolution de problème complexes peuvent utiliser des cartes physiques dédiées, telle un GPU pour le calcul massivement parallèle ou le réseau pour un service de stockage. Il devient alors nécessaire de prendre en compte un modèle de consommation considérant les activités de plusieurs ressources. Nous avons adapté le modèle de consommation linéaire pour considérer plusieurs ressources arbitraires. Cette adaptation a produit un modèle où l'activité de chaque ressource considérée a un impact linéaire sur la consommation du serveur.

Notre modèle linéaire multi-ressources associe à chaque ressource r d'un serveur s une consommation maximale $P_r(s)$ induite par l'utilisation exclusive de cette ressource sur ce serveur. Dans une configuration virtuelle, la consommation du serveur induite par l'activité de cette ressource est proportionnelle à cette activité, et donc aux activités des VM que ce serveur héberge.

La consommation d'un serveur s sous un tel modèle est alors définie comme

$$P(s) = P_{min}(s) + \sum_{resource\, r} charge_r(s) \times P_r(s)$$

Où $charge_r(s)$ est la charge normalisée de la ressource r sur le serveur s, donc valant au maximum 1 à pleine charge ; $P_{min}(s)$ la consommation du serveur sans aucune activité

sur ces ressources ; et $P_r(s)$, comme indiqué plus haut, la consommation maximale induite par l'utilisation exclusive de la ressource r sur le serveur s.

Ce modèle a l'avantage d'être plus expressif que le modèle linéaire en CPU, qui est de fait un cas spécifique de consommation linéaire. En particulier, il permet de considérer dans un même problème des modèles de consommation des serveurs considérant chacun des ressources différentes.

Un tel modèle nécessite cependant, pour être utilisé, de connaître l'activité de chaque ressource utilisée dans le serveur modélisé. Cette connaissance peut être difficile à obtenir quand les fonctionnalités du SGIV sont réduites. De plus, étant plus complexe que le modèle linéaire en CPU, ce modèle demande l'ajout de contraintes supplémentaires lors de la résolution d'un problème de placement et est donc moins performant. Comme décrit en 4.3.2, il y a néanmoins des cas où le modèle linéaire en CPU n'est pas suffisant pour décrire la consommation d'un serveur.

Ce modèle linéaire permet, de même que le modèle linéaire en CPU, d'associer à chaque couple (VM, serveur) la consommation induite par la vm sur le serveur. La reformulation de la consommation totale vue précédemment peut alors être utilisée, prenant en compte les consommations induites par les VM sur les serveurs et l'allocation des VM sur chaque serveur. Contrairement au modèle linéaire en CPU, ce modèle peut produire un nombre important de contraintes si le nombre de ressources à considérer devient important. Dans ce cas, la reformulation par consommation des VM devient une alternative envisageable. En effet, le nombre de contraintes produites par cette reformulation est constant avec le nombre de ressources.

Les modèles linéaires permettent d'intégrer de nombreux travaux sur la consommation des serveurs, cependant d'autres modèles mono-ressource sont parfois présentés. En particulier, nous avons intégré un modèle quadratique et un modèle constant par paliers de CPU.

Consommation quadratique en CPU

Ce modèle, évoqué par exemple dans [UKIN10], définit la consommation d'un serveur comme étant une fonction polynomiale de degré deux de la charge CPU de ce serveur. Il est donc une extension du modèle linéaire, qui est un polynôme de degré un.

$$P_{quad,CPU}(serveur\ s) = \gamma + CPU(s) \times \beta + CPU^2(s) \times \alpha$$

Lorsque la charge CPU est normalisée, la charge CPU varie de 0 à 1 et donc la consommation de β à $\alpha + \beta + \gamma$. De même que pour la consommation linéaire, nous notons ces valeurs de consommation minimales et maximales P_{min} et P_{max}. Nous définissons ensuite le taux de linéarité $t_{lin} = \frac{\beta}{\alpha+\beta}$. Ce coefficient de linéarité vaut 0 si β vaut 0, et 1 si α vaut 0, c'est-à-dire lorsque le modèle est linéaire. En utilisant ces nouvelles

variables, nous avons alors

$$\alpha = (1 - t_{lin}) \times (P_{max} - P_{min})$$
$$\beta = t_{lin} \times (P_{max} - P_{min})$$
$$\gamma = P_{min}$$

Ces variables nous permettent d'intégrer des modèles de consommation quadratique issus d'observations des centres. En effet l'observation des valeurs de consommation maximale des serveurs ne nécessitent pas de travail spécifique. La déduction de taux de linéarité demande cependant l'analyse des traces d'activité.

Dans certains cas les modèles de consommation continus ne permettent pas de prendre en compte les variations de consommation des serveurs. Il faut alors utilise des modèles de consommation discrets, tel le modèle par paliers de CPU

Consommation par paliers en CPU

Quand les observations de consommation des serveurs sont faites sur des valeurs de charge CPU, les modèles linéaires peuvent s'avérer trop imprécis. En particulier, certains serveurs embarquant plusieurs CPU avec chacun plusieurs cœurs CPU peuvent n'activer qu'un nombre réduit de ces cœur, maintenant les processus exécutés sur le nombre restreint de cœur actifs. Dans certains cas, la puissance électrique consommée par un cœur est quasiment constante quand ce cœur est activé. Dans ce cas la consommation d'un serveur est fonction non pas directement de la charge CPU des cœurs mais plutôt du nombre de cœurs activés, bien que ce nombre dépende de la charge CPU des cœurs. Nous avons développé un modèle de consommation constante par pallier de charge CPU qui permette de s'approcher de ces modèles.

Ce modèle associe à différents paliers (*eg.* de 0 à 20%, puis de 21 à 40%) de charge CPU une consommation constante du serveur. Il peut être plus précis dans certains cas qu'un modèle linéaire, mais demande l'injection de contraintes de seuil dans le problème, qui ont un impact important sur les performances.

Un intérêt supplémentaire à ce modèle est la prise en compte des seuils d'efficacité optimale. Le seuil d'efficacité optimale d'un serveur s, soumis à un modèle de consommation électrique en lien avec sa charge CPU, est la valeur de charge CPU pour laquelle le ratio d'efficacité est maximisé, avec la formule

$$eff_{CPU}(s) = CPU(s)/P(s)$$

Ce seuil indique à partir de quel charge CPU du serveur il n'est plus intéressant, d'un point de vue énergétique, d'ajouter des VM sur ce serveur car son efficacité énergétique va décroître.

Dans le cadre d'un modèle linéaire où $P(s) = \beta + \alpha \times CPU(s)$, cette efficacité vaut

$$eff_{lin,CPU}(s) = \frac{CPU(s)}{\beta + \alpha \times CPU(s)} = \frac{1}{\alpha}(1 - \frac{\beta}{\beta + \alpha \times CPU(s)})$$

qui est strictement croissante avec la charge CPU de s, impliquant qu'un serveur ayant une consommation linéaire en CPU sera toujours plus efficace à pleine charge. Ce qui n'est pas toujours le cas pour des serveurs réels, par exemple [SPE08] indique que l'efficacité optimale du modèle ALTOS R380 F2 se situe aux alentours de 70% de charge CPU, la montée à une charge de 100% réduisant l'efficacité énergétique de ce serveur de 10%.

Le modèle par paliers CPU permet, contrairement au modèle linéaire en CPU, de spécifier un seuil d'efficacité optimale différent de 1 pour le CPU (*eg.* 70% au lieu de 100%). Cette donnée statistique permet de trier plus finement les serveurs par efficacité énergétique, et donc d'approcher plus rapidement d'une solution optimale. Cependant, ce modèle reste coûteux en terme de mémoire et de performances de résolution du problème, et ne devrait être utilisé que quand la modélisation linéaire n'est pas satisfaisante.

Ces quelques modèles nous permettent d'intégrer les consommations des serveurs dans un problème de placement, et par la suite d'exprimer des besoins, sous forme de coûts et de règles, demandés par l'administrateur.

4.3.3 Gestion d'un centre sous contrainte énergétique

L'administrateur d'un centre ne connaît pas le processus de résolution d'un problème de placement. Son interaction avec OPTIPLACE se limite à spécifier des règles à respecter dans le centre ainsi qu'un éventuel objectif à prendre en compte. Ainsi, une fois la vue énergétique paramétrée, l'administrateur utilise cette vue pour spécifier des règles et objectifs liés à la consommation des serveurs.

Consommation des serveurs

Le premier objectif que l'administrateur peut considérer est la réduction de l'empreinte énergétique des serveurs. Cette empreinte énergétique dépend principalement des modèles de serveurs et de la configuration virtuelle du centre, mais aussi de la capacité du centre à éteindre des serveurs inutilisés.

Quand l'administrateur spécifie cet objectif, la fonction de coût associée est trivialement la somme des consommation des serveurs. Si les serveurs peuvent être éteints, cette somme doit prendre en compte la consommation avec extinction des serveurs, qui est la consommation du serveur quand celui-ci héberge au moins une VM et 0 quand aucune VM n'est hébergée sur le serveur.

Sans information spécifique des modèles de consommation utilisés, nous avons donc la consommation du centre c, contenant des serveurs (s)

$$P(c) = \sum_{s \in c} (P(s) \times h(s)$$

avec $P(s)$ la puissance du serveur s ; $h(s)$ la valeur booléenne valant 0 ssi aucune VM n'est hébergée sur le serveur s et que l'administrateur peut éteindre ce serveur.

Si par contre les consommations des serveurs sont toutes linéaires, nous pouvons utiliser la reformulation décrite en 4.3.2

$$P_{lin}(c) = \sum_{s \in c} P_{min}(s) \times h(s) + \sum_{vm \in c} P_{\delta}(vm)$$

qui utilise une plus grande quantité de mémoire mais permet de résoudre plus tôt le problème de recherche d'optimal.

L'utilisation de la reformulation linéaire et l'extinction des serveurs inutilisés sont deux paramètres de la vue énergétique, désactivables par le développeur.

Équilibrage énergétique

Les problématiques de flux thermiques dans le centre, et particulièrement les problématiques de points chauds, peuvent être analysés de différentes manières. Par exemple, la présence de points chauds dans un centre peut être expliquée par un mauvais réglage du thermostat du système de refroidissement, par la présence de matériel gênant les flux d'air dans le centre, par des phénomènes de circulation parasites de l'air, mais aussi par une trop importante densité énergétique au sein de certaines parties du centre.

Ce dernier problème peut être considéré dans la vue énergétique en équilibrant les charges énergétiques des serveurs. En effet, certains serveurs (en particulier les "serveurs lames" , des regroupements d'unité de calcul) ont une consommation électrique bien plus importante que les serveurs traditionnels du fait de leur orientation HPC.

Bien que de tels serveurs puissent avoir très bonne efficacité énergétique du CPU (définie en 4.3.2), une utilisation intense de leur CPU implique un pic local de consommation électrique et donc de production de chaleur. Lorsque le système de refroidissement est mutualisé entre les serveurs, ce pic de chaleur localisé est équilibré par une augmentation globale du débit d'air froid dans le centre. Le système de refroidissement produit alors un travail supplémentaire dont l'intérêt est limité à un faible nombre de serveurs. Dans ce cas, le gain d'efficacité apporté par les serveurs à haute densité énergétique est annulé par l'augmentation de la consommation du système de refroidissement.

OPTIPLACE permet d'équilibrer la consommation entre les serveurs les plus énergivores, réduisant ainsi les pics de consommation locaux. Cette demande d'équilibrage

parmi les serveurs d'un groupe g se traduit comme la réduction de l'écart de consommation entre les serveurs de ce groupe.

$$\delta_P(g) = max_{s \in g}(P(s)) - min_{s \in g}(P(s))$$

Cette fonctionnalité est cependant limitée à un centre mélangeant des serveurs très énergivores et des serveurs à faible consommation. Cependant, le problème thermique peut aussi être abordé sous un autre angle, sous la condition d'avoir accès à des données supplémentaires.

La vue hotspot

Le problème de la recirculation de l'air chaud dans les centres virtualisés, pris en compte par exemple dans [TGV07], réduit l'efficacité du système de refroidissement de l'air d'un centre. Si les modèles de flux thermiques sont très compliqués à mettre en œuvre, il a été montré que le placement des VM sur les serveurs les plus proches de l'entrée de l'air froid réduisait l'effet de ce problème.

La vue *hotspot* est une extension de la vue énergétique, utilisant la notion de distance d'un serveur à une entrée d'air froid. Typiquement, dans un centre disposé en allées chaudes et froides, cette notion de distance correspond à la hauteur à laquelle est placé un serveur dans son allée.

Cette vue propose une règle de consommation énergétique, demandant à ce que les serveurs les plus distants d'une entrée d'air froid aient une charge énergétique plus faible que celle des serveurs plus proches. Cette règle peut être paramétré par un groupes de serveurs sur lesquels elle s'applique, et en interne assure la relation

$$\forall(s1,\, s2) \in (g \times g) \,:\, dist(s1) > dist(s2) \implies P(s1) \leq P(s2)$$

Bien que cette vue soit basée sur un modèle simplifié, elle montre que la vue énergétique peut aussi être utilisée pour produire d'autres vues utilisant les notions de consommation des serveurs.

Power capping

Le fonctionnement d'un centre nécessite un apport fiable et important en électricité. Cet apport est satisfait par un contrat établi avec un fournisseur dédié, spécifiant les tarifications en vigueur. En particulier ce contrat indique une consommation maximale en dehors de laquelle il n'est plus garanti, ou pour un coût prohibitif.

Dans le cas où la consommation du centre excède cette consommation maximale, cet excès peut avoir différentes implications. Si un dépassement n'est pas prévu par ce contrat, le centre peut faire face à une défaillance électrique, pouvant générer des extinctions ou dans le pire des cas la destruction de matériel, par exemple par coupure du

système de refroidissement. Si le dépassement est prévu dans le contrat, il est généralement associé à une pénalité tarifaire réduisant la rentabilité du centre. Dans tous les cas, l'administrateur doit prendre en compte cette limite de consommation.

L'architecture et la consommation d'un centre sont soigneusement calculées à la construction de celui-ci. Cependant l'évolution rapide du matériel informatique, et en particulier de la densité de consommation des serveurs (exprimée en W/m^3) peut rapidement rendre ces données initiales obsolètes. Lorsque la consommation du centre devient trop importante, l'administrateur doit prendre des mesures pour la limiter.

De telles mesures peuvent être physiques, par l'extinction de matériel informatique : les serveurs inactifs, les systèmes redondants peuvent être éteints à un coût de performances très faibles. Par la suite les serveurs actifs peuvent aussi être éteints, mais leur extinction entrainera la perte des VM présentes, pouvant induire des pénalités selon les SLA des VM.

Ces mesures peuvent être mécaniques, par réduction de la consommation électrique des éléments d'un serveur. Le DVFS permet par exemple de réduire la consommation du CPU d'un serveur bien que réduisant aussi grandement ses performances. Certains modèles de serveurs permettent même de désactiver dynamiquement les cœurs CPU ou GPU installés, ainsi que les cartes réseaux redondantes d'un serveur.

Ces mesures peuvent enfin être basées sur la gestion des ressources virtualisée du centre. Ainsi les SGIV peuvent restreindre la consommation CPU, réseau, disque des VM, réduisant ainsi la consommation électrique des serveurs hôtes.

Dans tous les cas ces mesures s'accompagnent d'un coût, que ce soit en terme de sécurité des applications, de leurs performance, ou de la consommation totale du centre.

Le *Power Capping* permet à l'administrateur du centre de borner la consommation totale des serveurs. Cette règle n'a qu'un impact réduit sur la consommation du centre, car OPTIPLACE ne dispose que des actions de migration pour la gestion de ce dernier. Cependant, de par l'architecture modulaire d'OPTIPLACE, cette règle peut être utilisée dans d'autres vues. Il est ainsi possible dans une vue *performance*, exprimant la relation entre les performances d'un serveur et sa consommation électrique, de réduire la consommation de certains serveurs pour rester dans les limites spécifiées par cette règle.

4.3.4 Conclusion et ouverture

Grâce à notre architecture, nous sommes capables d'activer et désactiver au besoin la vue énergétique. Cette modularité nous permet de nous appuyer sur différents travaux déjà effectués sans devoir les ré-implémenter dans le cœur initial. Ainsi nous pouvons activer les vue HD et la vue énergétique en même temps.

Ce système permet le développement de nouvelles vues. En particulier un projet en cours propose d'intégrer différentes données électriques. Cette vue permettra de spécifier les phases utilisées par le serveurs, pour par exemple faire de l'équilibrage de phases,

les onduleurs utilisés pour améliorer les réponses aux pannes électriques et maximiser la disponibilité du centre.

Une autre vue en cours d'élaboration est la vue thermale, qui ajoute des informations de température en sortie des serveurs.

Si la PPC permet de modéliser et assembler facilement des notions et contraintes différentes, elle a comme défaut principal de rechercher une solution par le parcours d'un arbre de recherche. Un tel parcours pouvant être très coûteux, il est indispensable dans de tels outils de développer des heuristiques de recherches efficaces, et ce particulièrement quand on cherche à résoudre des problèmes impliquant plus de 1000 serveurs. D'un point de vue architectural, nous avons ajouté à OPTIPLACE la notion de stratégies de recherche ou heuristiques et des paramétrages spécifiques aux performances. Nous détaillons ces deux points dans la section suivante.

4.4 La recherche de solutions

La vue énergétique intègre les modèles de consommation des serveurs dans un problème de placement. Cependant, la PPC requiert autant de travail pour la déduction d'une stratégie de recherche que pour la modélisation du problème.

En effet, sans contrainte particulière, un problème de placement de n VM sur m serveurs propose m^n solutions potentielles. Bien sûr, les contraintes ajoutées, en particulier celles de ressources, réduisent le nombre de solutions au problème en interdisant le parcours d'une partie de l'arbre de recherche. De même, l'utilisation d'une variable de coût dans le problème ajoute des contraintes supplémentaires à chaque solution trouvée, et donc réduit l'espace des solutions potentielles. Cependant, ces contraintes ne sont pas suffisantes pour permettre le parcours d'un arbre de grande taille en temps réaliste. En effet, si un problème de placement de 30 VM sur 10 serveurs permet un parcours en une seconde, un problème manipulant 1000 serveurs et 3000 VM requiert alors plusieurs heures de calcul.

La résolution d'un problème par PPC considère non seulement le modèle du problème mais aussi l'exploration des solutions de ce problème, à travers des stratégies de recherches, ou heuristiques. Cette notion est directement accessible dans OPTIPLACE. Nous avons donc développé dans la vue énergétique des heuristiques dédiés à la réduction de la consommation énergétique des serveurs.

4.4.1 Différents types d'heuristiques

Dans notre cas d'étude, les heuristiques peuvent être classés en deux catégories, selon l'intégration de leur stratégie de recherche dans le processus de résolution du problème.

Les heuristiques dites *statiques* établissent une stratégie de recherche uniquement déduite des données initiales du problème. Elles effectuent une analyse à priori de ce

problème, en déduisent une solution intéressante, et tentent d'approcher cette solution durant l'exploration de l'arbre de recherche. De tels heuristiques vont typiquement ordonner les variables par des critères spécifiques, et lors d'un branchement dans le problème sélectionner les variables et leurs valeurs en se basant sur ce tri. Ces heuristiques sont optimistes, dans le sens où ils ignorent totalement les contraintes supplémentaires qui peuvent être appliquées au problème. En effet, les contraintes du problème peuvent rendre inatteignables les solutions déterminées par une analyse à priori, et donc faire perdre son sens à l'heuristique.

Les heuristiques dits *dynamiques* établissent progressivement leur stratégie de recherche en fonction de l'état d'exploration du système. Si leurs critères sont fixés de manière statiques, leurs choix sont dictés par une analyse dynamique du problème, selon la position dans l'arbre de recherche et les modifications apportées au système par les contraintes externes. Ces heuristiques dynamiques peuvent soit recalculer régulièrement l'analyse du problème durant le parcours des solutions, soit déléguer au solveur du problème la construction et la maintenance de cette stratégie.

Si une telle heuristique recalcule régulièrement sa stratégie de recherche, cette opération peut demander une surcharge CPU importante. Par exemple, le tri régulier des éléments du problème est une opération coûteuse en CPU, et ce d'autant plus que le nombre d'éléments à trier est important. Des stratégies de recherche peuvent n'effectuer que des tris partiels pour réduire cette surcharge CPU, mais la simple observation du système pour déduire la nécessité d'un tri peut être coûteuse.

Il est alors préférable d'utiliser les mécanismes présents dans le solveur pour ne recalculer les analyses que lorsque leur besoin se fait sentir. L'interface du solveur est alors utilisée pour injecter dans le problème un modèle d'analyse, dont les variables seront mis à jour automatiquement par le solveur durant le parcours de l'arbre de recherche. L'utilisation de ces variables permet alors de décider des actions à entreprendre durant le parcours de l'arbre de recherche. Cependant, de telles stratégies de recherche peuvent être très complexes à mettre en œuvre. En effet, cette injection peut nécessiter l'utilisation de structures de données complexes dans le solveur, par exemple une structure de collection triée.

De ce fait, nous nous sommes concentrés sur le développement d'heuristiques statiques, qui analysent par avance le problème de placement des VM sur les serveurs et les modèles énergétiques de ces serveurs. Ces heuristiques sélectionnent à chaque nœud de l'arbre de résolution la branche qui semble s'approcher le plus d'une solution optimale.

4.4.2 Approches mono-ressource

Les modèles les plus présents dans la littérature font apparaître la consommation d'un serveur comme principalement imputable à sa charge CPU. Dans certains cas cette ressource est la seule prise en compte dans les modèles. Il est donc justifié d'étudier d'abord

des heuristiques qui ne prennent en compte que des modèles de consommation basés sur une seule ressource, avant de les étendre à des modèles plus complexes.

Nous commençons donc par étudier un centre où les serveurs ont tous une consommation qui ne dépend que de l'activité d'une seule ressources, en l'occurrence le CPU. Étant donné la capacité d'OPTIPLACE à considérer de manière homogène les ressources consommées par les VM sur les serveurs, nous pouvons facilement remplacer le CPU par une autre ressource, *eg.* le GPU ou l'activité réseau. Cette classe de sous-problème permet d'obtenir de premiers résultats et de formaliser certaines notions.

Nous définissons tout d'abord l'efficacité d'un serveur s dont la consommation $P(s)$ n'est imputable qu'à l'utilisation d'une de ses ressources. Cette efficacité est définie, à une charge CPU $CPU(s)$ donnée, comme le quotient de cette charge CPU par la consommation de ce serveur à cette charge.

$$eff_{CPU}(s) = \frac{CPU(s)}{P(s)}$$

Cette fonction d'efficacité atteint une valeur maximale que nous notons $\overline{eff}_{CPU}(s)$.

$$\overline{eff}_{CPU}(s) = max_{CPU(s)}(eff_{CPU}(s))$$

Nous supposons que cette valeur n'est atteinte qu'en une valeur de consommation CPU du serveur que nous notons $\overline{CPU}_{eff}(s)$. Si cette efficacité maximale est atteinte pour différentes valeurs d'activité CPU, nous choisissons arbitrairement la plus grande de ces valeurs.

Nous pouvons ensuite comparer les différents serveurs par leurs valeurs d'efficacité maximale. Afin que ces comparaisons aient du sens, il faut tout d'abord que chaque serveur considéré ait un modèle de consommation basé sur son activité CPU. Il faut ensuite que les valeurs d'activité CPU associées à ces serveurs soient dans la même unité, ce qui est un requis pour OPTIPLACE, et interdit d'utiliser par exemple une charge normalisée ou un pourcentage de charge CPU.

Si le problème le permet, il est intuitivement plus intéressant de placer les VM sur les serveurs à plus haute efficacité possible, de manière à atteindre pour chaque serveur allumé son activité CPU qui maximise cette efficacité $\overline{CPU}_{eff}(s)$. Cette démarche correspond dans un problème de placement à l'allocation des VM sur les serveur à la plus grande efficacité maximale d'abord, jusqu'à atteindre leur efficacité maximale.

Cette démarche considère que les serveurs peuvent être chargés exactement à une valeur de CPU donnée afin d'atteindre leur efficacité optimale. Cette propriété ne peut être cependant assuré dans un problème réel, par exemple quand toute VM a une activité CPU supérieure à ce seuil d'efficacité maximale. Cette approche optimiste permet cependant de définir une heuristique de recherche simple.

Efficacité d'une consommation linéaire en CPU

Un cas particulier de la consommation mono-ressources est celui où un serveur a une consommation linéaire en CPU comme vu en 4.3.2, avec $P(s) = \alpha \times CPU(s) + \beta$ Dans ce cas la formule d'efficacité énergétique du serveur devient

$$eff_{CPU}(s) = \frac{CPU(s)}{\alpha \times CPU(s) + \beta} = \frac{1}{\alpha} \times (1 - \frac{\beta}{\alpha \times CPU(s) + \beta)}$$

Cette valeur d'efficacité est alors strictement croissante pour α et β strictement positifs. L'efficacité du serveur croît donc avec sa charge CPU, pour une valeur maximale à pleine charge CPU.

Pour un serveur à consommation électrique linéaire avec sa charge CPU il est donc intéressant de charger au maximum de sa capacité CPU ce serveur. De tels serveurs dans un même centre peuvent donc être comparés entre eux par leur efficacité à pleine charge.

Cependant, dans le cas où le centre n'est pas composé que de serveurs à consommation linéaire, l'efficacité des serveurs doit être calculée d'une autre manière pour comparer les serveurs entre eux

Efficacité d'un modèle de consommation mono-ressource générique

Dans le cas où la consommation d'un serveur n'est pas linéaire avec une ressource, il faut déterminer pour chaque serveur sa valeur d'activité CPU maximisant cette efficacité. L'interface générique d'un modèle de consommation implémente une telle fonctionnalité.

La fonctionnalité retenue permet de déterminer l'efficacité d'un serveur relative à un une VM type. Un modèle de consommation indique donc, pour son serveur et le profil de VM pris en compte, le nombre de VM n à héberger sur ce serveur pour maximiser l'efficacité de ce serveur, avec

$$eff_{vm}(s, n) = \frac{n}{P(s, n \times vm)}$$

Ainsi, en spécifiant une VM ne consommant que 1 CPU, un modèle renvoie le seuil d'efficacité maximale du serveur $\overline{CPU}_{eff}(s)$ définit précédemment.

Cette fonctionnalité est implémentée par défaut dans les modèles de la manière suivante : D'abord, le modèle détermine le nombre maximum de VM qui peuvent être hébergées sur le serveur. Ensuite, le modèle évalue l'efficacité du serveur pour de 1 à ce maximum de VM hébergées, et retient l'efficacité maximale ainsi que le seuil associé. Cette implémentation est modifiable par les développeurs, par exemple le modèle linéaire renvoie le nombre maximum de VM que le serveur peut héberger.

Une heuristique peut ainsi utiliser des modèles génériques pour déduire, en spécifiant un profil de VM, les efficacités de chaque serveur.

Heuristique de placement

Les premières heuristiques développées utilisent cette notion d'efficacité maximale des serveurs, dans un cadre de consommation mono-ressource. Elles placent en priorité les VM sur le serveur à la plus importante efficacité maximale, dans le but de le faire atteindre cette efficacité. Pour cela elles utilisent une liste des serveurs triée par leur efficacité maximale décroissante.

Une première heuristique déplace les VM des serveurs les moins efficaces vers le serveur le plus efficace. Pour cela nous définissons l'efficacité initiale d'un serveur comme étant son efficacité dans la configuration initiale. Cette heuristique utilise une deuxième liste des serveurs triée par leur efficacité initiale croissante. Lors de l'exploration de l'arbre de recherche, cette heuristique sélectionne les VM des serveurs à plus basse efficacité initiale et les place sur les serveur à plus haute efficacité maximale.

Cependant, les solutions obtenues par cette heuristique déplacent toutes les VM présentes sur le serveur à plus haute efficacité maximale, pour les remplacer par des VM des serveurs les moins efficaces. En particulier, quand la configuration de base était optimale du point de vue énergétique, cette heuristique produit un nombre important de migrations sans améliorer la consommation du centre.

L'heuristique suivante se focalise non plus sur la migration des VM des serveurs initialement inefficaces, mais plutôt sur l'utilisation au maximum des serveurs à plus haute efficacité maximale, en utilisant les mêmes liste que l'heuristique précédente. Durant l'exploration de l'arbre du problème, elle sélectionne d'abord le serveur le plus efficace disponible pour lui allouer des VM. Elle tente ensuite de placer sur ce serveur, d'abord les VM qu'il hébergeait dans la configuration initiale afin de réduire le nombre de migrations, puis les VM des serveurs les moins efficaces dans la configuration initiale.

Cette deuxième heuristique réduit le nombre de migrations nécessaires pour atteindre la configuration solution, cependant elle ne prend pas en compte les contraintes liées au packing des ressources. En effet ces contraintes peuvent limiter les activités maximales des serveurs. Par exemple si la charge mémoire moyenne des serveurs est bien plus importante que leur charge CPU moyenne, il est peu probable que ces serveurs puissent atteindre leur charge CPU maximale. Dans des centres réels, ce cas peut se présenter par exemple pour des services de stockage à faible activité, où les réservations mémoire sont importantes quand les activités CPU sont faibles. Dans ce cas une heuristique dynamique, prenant en compte la charge CPU maximale des serveurs ainsi que leur consommation maximale serait plus appropriée.

Malgré leurs limites, ces heuristiques permettent de réaliser des premières évaluations de performance de la vue énergétique. La notion d'efficacité énergétique d'un serveur est de plus utilisée dans les modèles à plusieurs ressources.

4.4.3 Approche multi-ressources

Dans le cas où les activités de plusieurs ressources sont prises en compte par le modèle de consommation, il n'est plus possible d'utiliser la notion d'efficacité énergétique d'un serveur. En effet, la charge d'un serveur devient un vecteur multi-dimensionnel et non plus une valeur dans IN. Il n'est donc plus possible de comparer, et trier, les efficacités maximales et minimales d'un serveur sans redéfinir cette notion de charge.

Une approche simple consisterait à définir une charge réduite des serveurs en utilisant leurs vecteurs d'activité des ressources. Cette approche est moins simple qu'il n'y parait. Comme nous voulons comparer les serveurs entre eux, la formulation de cette charge doit être la même pour chaque serveur. Par exemple, nous pourrions effectuer une somme pondérée des activités sur chaque serveur, mais cela demande une pondération qui représente cette notion d'efficacité.

Comparaison des serveurs par profil type

Cette relation utilise un modèle de VM type, dont les activités servent de base à la notion d'efficacité des serveurs. Étant donné que les activités des VM sont connues, nous pouvons nous baser sur l'activité moyenne d'une VM pour établir un profil moyen de VM du centre. Les activités d'un serveur sous une charge n donnée correspondent alors aux activités produites par n VM de ce type. Cette comparaison est possible grâce à la possibilité pour chaque modèle de calculer son efficacité maximale relative à une VM pour un serveur donné, $\overline{eff}_{vm}(s)$. Typiquement, une telle relation est linéaire et permet une continuité avec la formule d'efficacité vu précédemment, quand une seule ressource entre en compte dans le modèle de consommation.

Une fois cette VM type déterminée, les serveurs sont triés par leur efficacité maximale décroissante. Cependant, dans le cas où les VM ont des activités très hétérogènes, ce profil de charge moyenne perd de son intérêt. En effet, les serveurs auront alors une efficacité maximale relative à une VM très différente selon les VM considérées. Il faut alors prendre en compte ces différences d'efficacité.

Approche par comparaison monogame

Plutôt que de comparer les efficacités des serveurs de manière absolue, cette approche calcule les affinités entre chaque VM et chaque serveur. Pour cela, elle calcule la matrice d'efficacités monogame, qui à chaque couple (VM v, serveur s) associe l'efficacité maximale de s relative à v telle que vue précédemment.

$$EFF_{mono}(s, v) = \overline{eff}_v(s)$$

Nous nommons cette approche une comparaison monogame car les valeurs d'affinité sont calculées en supposant que chaque serveur n'exécute que des VM de même profil.

Cette approche prend son sens quand les VM du centre peuvent être regroupées par activités semblables. Elle peut être améliorée pour prendre en compte des ensembles de VM, réduisant l'empreinte mémoire nécessaire.

Cette matrice des efficacités monogames permet d'obtenir, à une VM v fixée, le vecteur ligne des efficacités des serveur relativement à cette VM. Ce vecteur ne permet pas de comparer directement les VM entre elles. En effet, même sur un seul serveur, les valeurs d'efficacité maximale présentent dépendent des activités de chaque VM et non uniquement de leur répartition.

Par exemple, prenons un serveur s, de capacité 16CPU et dont la consommation varie linéairement de 0 à 4W avec son activité CPU. Considérons deux VM v1, utilisant 1 CPU, et v2, utilisant 2 CPU. Nous savons que s peut héberger au maximum 16 VM du type v1, ou 8 du type v2, et son efficacité maximale est atteinte à pleine charge, donc à 4W de consommation. Nous avons alors, $\overline{eff}_{v1}(s) = 16/4 = 4$ et $\overline{eff}_{v2}(s) = 8/4 = 2$. Ces valeurs d'efficacité sont différentes, cependant l'hébergement de 16 v1 ou 8 v2 produit exactement la même activité et la même consommation, c'est-à-dire l'efficacité du serveur eu égard à ses activités devrait être la même.

De fait, ces valeurs d'efficacité des VM sur les serveurs ne peuvent pas être comparées directement. Nous comparons à la place les VM par l'amplitude de ces valeurs. Pour cela, nous déterminons, à une VM v, les maximum et minimum d'efficacité monogame de cette VM dans le centre. Le rapport de ces valeurs nous donne le potentiel d'efficacité de v.

$$eff_{pot}(v) = \frac{max_s(\overline{eff}_v(s))}{min_s(\overline{eff}_v(s))}$$

Ces valeurs d'efficacité potentielle nous permettent de déterminer les priorités des VM. En effet, si une VM a un fort potentiel d'efficacité cela signifie que l'efficacité du serveur de cette VM peut grandement varier. Ainsi, une telle VM doit être allouée prioritairement par rapport à, par exemple, une VM à potentiel d'efficacité de 1, c'est à dire que l'efficacité monogame de cette VM est constant quel que soit le serveur considéré.

Ces valeurs nous permettent de paramétrer L'heuristique suivante. Elle consiste à sélectionner les VM à placer par ordre décroissant de potentiel d'efficacité. Les VM sélectionnées sont hébergées sur le serveur qui leur permet une plus grande efficacité monogame maximale. En cas d'égalité des serveurs nous choisissons celui qui héberge la VM, puis le serveur le plus chargé en ressources.

Cette heuristique souffre cependant d'un problème de demande de migration trop importante, en cela qu'elle ne cherche pas à maintenir en priorité les VM sur les serveurs déjà chargés. Elle n'est alors pas utilisable quand l'administrateur souhaite répondre en temps réel à des variations d'activité des VM ou décider en temps réel où placer une nouvelle VM. En revanche, si l'optimisation est lancée par exemple une fois par jour elle est tout à fait adaptée.

Cette heuristique n'a pas été intégrée à OPTIPLACE et nécessite des recherches supplémentaires. Cependant, elle nous permet de déterminer les informations que les modèles de consommations doivent fournir pour permettre des heuristiques complexes liés à la réduction de la consommation dans un environnement hétérogène.

4.5 Conclusion

Nous avons apporté dans OPTIPLACE la modularité dont ENTROPY manquait pour permettre le développement de modules de contraintes externes. Cette modularité a nécessité la réduction des fonctionnalités de base dans OPTIPLACE, mais permet de développer plus facilement des contraintes, objectifs et heuristiques de recherche dédiés à des problèmes spécifiques.

En utilisant cette modularité, nous avons développé une vue énergétique considérant les consommations électriques des serveurs du centre. Cette vue utilise des modèles génériques de consommation axés sur les activités ressources des VM. Elle permet de focaliser la recherche de solution vers une réduction de l'empreinte énergétique du centre ainsi que de borner ou d'équilibrer les consommations des serveurs. Nous avons proposé plusieurs heuristiques selon la complexité du problème considéré pour obtenir une réduction de la consommation du centre.

Les tests que nous avons effectués et détaillés au chapitre 6 indiquent des temps de recherche de l'ordre de la minute pour des centres de 1000 serveurs. Bien que ces tests fournissent des solutions approchées, et non optimales comme le faisait ENTROPY, ces valeurs sont très proches de l'optimale lorsque les centre sont suffisamment simples.

Notre système est très flexible, cependant les évaluations ont été réalisées sur des centres théoriques très simples et un centre réel est nécessaire pour une évaluation pratique du système. Nous pouvons alors simuler un centre complet ou bien manipuler un centre réel. Étant donné que OPTIPLACE nécessite une première passe de modélisation du centre, il ne nous parait pas raisonnable d'utiliser ces modèles approchés comme critères d'évaluation. De plus nous n'avons connaissance d'aucun système d'injection d'activité système qui soit distribué et reproductible dans un centre de grande taille, d'ou le développement de StressCloud, détaillé au chapitre suivant.

5

StressCloud

Sommaire

STRESSCLOUD est un outil d'injection d'activités au sein de VM hébergées sur un centre virtualisé. Ces activités sont visibles au niveau du système d'exploitation des VM, et sont aussi appelées des *charges système*.

Les injections d'activité sont contrôlées et synchronisées par un gestionnaire centralisé en vue d'exécuter des scénarios d'activités reproductibles. Les activités système utilisent les ressources mises à disposition des VM par leur serveur hôte. Les fonctionnalités de cet outil incluent la sélection des VM participant à un scénario, la spécification

et la planification de leurs activités ainsi que l'observation des performances des VM en réponse à ces activités.

5.1 Problématique

L'administration d'un parc informatique de serveurs est une tâche complexe. Celle-ci est généralement confiée à un administrateur système dont le rôle consiste non seulement en l'installation et la maintenance du parc, mais aussi en la bonne exécution des services hébergés par le parc malgré des contraintes fluctuantes. Bien évidement, plus le nombre d'éléments à administrer est important, plus la tâche d'administration devient complexe.

La virtualisation a largement contribué à la complexification de cette tâche, car au delà des simples aspects techniques, elle a multiplié les éléments à surveiller et administrer dans un parc. Ces dernières années l'augmentation du nombre d'éléments a accru le besoin de s'appuyer sur des gestionnaires d'administration pour les centres de grande taille. En particulier, les gestionnaires de centre virtualisé permettent d'observer un parc de serveurs et VM, d'analyser des données de performances, de coût de fonctionnement, de sécurité, d'en déduire des tâches d'administration pour améliorer ces indicateurs et de les appliquer de manière plus ou moins automatique. Ces tâches d'administration peuvent modifier l'état du centre, par la création et l'allocation des VM sur les serveurs, la gestion des priorités d'accès aux ressources sur ces serveurs, le déplacement de VM sur des serveurs plus efficaces, voire l'arrêt des VM causant un problème et l'extinction ou l'allumage de serveurs. De telles actions permettent par exemple de réduire le nombre de serveurs nécessaires au fonctionnement d'un parc de VM, afin de réduire la consommation électrique totale du centre. Ils peuvent aussi assurer le respect de contraintes de performances, augmenter la longévité du matériel, le tout en conservant des contraintes utilisateurs et administrateurs spécifiques.

La complexité de la gestion d'un tel centre augmente évidemment avec le nombre de serveurs et VM que celui-ci contient, mais également suivant les objectifs et les contraintes spécifiques au centre. Par exemple, une approche d'administration centrée sur la réduction des consommations des serveurs est antagoniste avec une demande de réservation d'un serveur par un utilisateur. Ces gestionnaires sont au centre de toutes les attentions, le choix d'un gestionnaire et de son paramétrage a un impact important sur le bon fonctionnement d'un centre virtualisé. Au-delà de notre première problématique liée au teste de la solution OPTIPLACE, il est indispensable de pouvoir tester et comparer différents gestionnaires et différents paramétrages.

Plusieurs approches permettent de valider et comparer des stratégies de gestion. La simulation d'un centre utilise des modèles issus d'observations du centre pour anticiper l'effet des stratégies de gestion. L'injection de charge quant à elle consiste à manipuler les services informatiques exécutés dans le centre pour y produire une activité spécifique, et y observer l'impact de cette activité sur le gestionnaire du centre.

La simulation permet de valider un gestionnaire sur des centres modélisés, mais plus la taille du centre augmente, et plus la modélisation nécessite un travail important. En effet, certains paramètres du centre, tels la consommation des serveurs, les flux thermiques dans le centre ou les temps de réponse aux requêtes, sont très complexes à modéliser et peuvent rendre la simulation impossible en pratique. Le paramétrage et la validation des modèles utilisés peuvent aussi être très difficiles à obtenir. En particulier chaque serveur, du fait de sa position dans le centre, de détails propres à sa fabrication, de ses paramètres de fonctionnement, peut suivre un modèle différent des autres serveurs du centre. Un administrateur préférera donc évaluer des stratégies de gestion sur un centre réel avec des charges plausibles plutôt que sur des modèles dont la validation nécessite une quantité très importante de travail.

L'injection de charge applicative dans un centre n'est pas non plus une tâche triviale [Gar07]. Elle demande d'abord la connaissance des services présents dans le centre, ce qui n'est pas le cas quand le centre met à disposition son infrastructure (modèle IaaS). Dans ces centres la sémantique des services hébergés est inconnue de l'administrateur dont la connaissance se limite, au travers du gestionnaire du centre, aux activités ressources des VM(utilisation du CPU, réservation de mémoire, activité réseau entre autres). Quand ces services sont connus, l'injection de charge doit pouvoir modifier l'activité de ces services. Pour cela il est nécessaire de connaître à minima le protocole d'interaction avec le service à charger, ce qui peut entraîner des développements spécifiques quand cette injection de charge n'a pas été prévue dans le service. En cas d'hébergement d'un service propriétaire ou non ouvert à l'interaction (pour secret industriel par exemple), cette injection de charge est même impossible. Enfin, en vue de reproduire une charge réaliste, cette injection requiert l'acquisition de traces d'activités de ces services en quantité et qualité suffisantes pour déterminer les paramètres d'injection.

Nous proposons une troisième approche pour l'évaluation de centre : l'exécution de scénarios d'activités système (ou charges système) basés non plus sur les services mais sur les accès aux ressources système dans un centre virtualisé. En effet, la plupart des gestionnaires n'ont pas accès à la sémantique des services hébergés par le centre [SMLF09]. Les VM sont alors considérées comme des boites noires, et les algorithmes de décision se basent uniquement sur les utilisations par les VM des ressources serveur. Dans notre contexte, il parait plus judicieux d'effectuer une injection de charge non plus applicative mais au niveau du système. Le terme de "charge système" ayant déjà un sens dans le domaine de l'administration système, nous appelons donc ces charge au niveau système des activités système. Des traces d'accès aux ressources sont communément observées dans les centres virtualisés, et ne nécessitent aucune connaissance des services présents dans le centre. Ces traces peuvent donc après traitement être injectées directement dans un centre réel sans avoir besoin de modéliser ce dernier.

La section suivante 5.2 présente un état de l'art des travaux apparentés dans le do-

maine de la validation des gestionnaires de IaaS. La section 5.3 détaille notre solution. Cette section est suivie en section 5.4 par une évaluation complète de l'outil proposé.

5.2 Travaux apparentés

L'évaluation d'une (de) politique(s) d'administration des SGIV est une tâche complexe. Dans de précédents travaux [HLM+09] portant sur la consolidation dynamique de VM pour la minimisation énergétique, les auteurs évaluaient principalement la politique d'administration via la simulation d'un centre de données. Utiliser un simulateur permet de tester et valider l'algorithme sur un grand nombre de configuration, mais ne permet pas de mettre en évidence certains effets de bords, comme la surcharge réseau liée aux multiples migrations des VM. Évaluer une politique d'administration sur infrastructure réelle permet d'être au plus proche de l'environnement final d'exécution et ainsi d'identifier et d'analyser d'éventuels effets de bord. Comme les algorithmes des SGIV s'intéressent principalement à l'utilisation des ressources d'un centre de données, leur évaluation ne peut se réaliser que sur une infrastructure réelle avec des VM consommant des ressources système.

Pour cela, il est nécessaire d'injecter une charge sur l'infrastructure et de vérifier le comportement de la politique d'administration du SGIV. L'injection de charge peut être faite à bas niveau (charge système) ou à haut niveau (charge de l'application). L'injection d'une charge à bas niveau consiste en l'utilisation contrôlée des ressources matérielles, telles le CPU, le disque dur ou la carte réseau, et permet par exemple de vérifier la stabilité du système sous cette charge. L'injection de charge à haut niveau correspond à l'utilisation contrôlée d'une application spécifique, par exemple en simulant sur un serveur web des requêtes HTTP provenant de plusieurs utilisateurs, permettant de s'assurer des performances du serveur sous des charges usuelles ou importantes. De ce fait, les injecteurs de charge à haut niveau sont spécifiques à un type d'application (HPC, Web, simulation bancaire etc.). Si le centre de données possède plusieurs types applicatifs, il sera nécessaire de déployer et maîtriser un grand nombre d'injecteurs de charge (RUBIs, NASGrid, Clif, etc.). Quand les injecteurs spécifiques à une application n'existent pas, il faut de plus les développer et les maintenir à jour.

Typiquement, Mistral [JHJ+10] est un système d'analyse des activités des VM et d'optimisation de la consommation du centre. Jung et al. évaluent leur algorithme sur un centre de données exécutant le benchmark RUBIs sur une architecture multi-tiers en local. Un serveur externe envoie des requêtes vers les serveurs de cette architecture. Ce benchmark permet de simuler des activités multi-tiers de VM, cependant il fonctionne au niveau de la couche logicielle, en simulant la réponse à des requêtes. Il n'est donc pas possible de manipuler des activité spécifiques des VM, car leur charge CPU dépend la fréquence d'exécution des requêtes reçues sur le réseau et réparties sur chaque tiers.

Or, les algorithmes des SGIV considèrent principalement les charges relatives aux

ressources du système : CPU, RAM, disque, réseau, etc.. Une injection de charge à haut niveau peut ne pas être satisfaisante : si cette injection produit bien une utilisation des ressources serveurs, la complexité des injecteurs ne permet pas de manipuler spécifiquement une de ces ressources, qui est une condition à l'évaluation des algorithmes IaaS. Enfin, comme évoqué les injecteurs de charge a haut niveau sont spécifiques à un type d'application (HPC, Web, simulation bancaire etc.) et l'utilisation d'un tel système suppose l'adaptation de ces injecteurs au système multi-tiers effectivement exécuté dans le centre.

Plusieurs solutions pour l'évaluation des SGIV ont été proposées dans la littérature. Par exemple, le système VGreen présenté dans [DMR10] par Gaurav Dhiman et al. récupère les informations d'activité des VM dans un centre et en déduit une réorganisation des VM sur les serveurs. Ce système est évalué sur un centre de test de trois serveurs et plusieurs VM, chaque VM exécutant un benchmark utilisant le CPU seul ou utilisant le CPU et la mémoire. Si ce système permet de spécifier les accès à effectuer aux ressources CPU et RAM, les VM de test ne peuvent pas être contrôlées après leur démarrage, leur gestion n'est pas centralisée, et il est difficile de changer les types de benchmark que chaque VM devra exécuter. L'adaptation de ce système à des charges spécifiques est donc impossible en l'état.

De même, Enacloud[LLH+09] est un système de gestion d'équilibrage des charges d'un centre dynamique, permettant de choisir le serveur hébergeant une tâche puis de migrer les tâches existantes selon leur activités. Il est testé sur un centre de 60 serveurs, sur lequel sont exécutées des VM contenant des benchmarks HPC, des applications de compilation ou des webservers, démarrées et arrêtées à des moments prédéfinis. Dans cet environnement d'évaluation, on ne peut donc pas contrôler finement les ressources système consommées.

Les travaux de [MWY+10] présentent un système de reconfiguration de centre par un algorithme génétique. Ce système est testé avec un centre de 300 serveurs et des VM exécutant un serveur web, vers lesquels sont envoyées des requêtes depuis un émetteur centralisé. Cette évaluation est proche des objectifs fixés, cependant la nécessité d'un émetteur de requêtes ne permet pas le passage à l'échelle, de plus les activités CPU et réseau de chaque VM sont fortement couplées. En effet le seul paramètre de l'évaluation est la fréquence d'envoi des requêtes, qui influe à la fois sur l'activité réseau et sur l'activité CPU des VM exécutées.

L'approche à notre sens la plus intéressante reste l'injecteur de charge CLIF présenté dans [Dil08] et JStress présenté dans [K W07], qui utilisent des injecteurs de charge pour modifier l'activité des VM. Cependant ces approches nécessitent d'exécuter sur le centre un type de VM pour chaque type de charge que l'on veut injecter.

Tous ces travaux présentent des évaluation de leurs solutions, mais à notre connaissance il n'existe pas encore de système travaillant au niveau IaaS et permettant à la fois de contrôler finement la consommation des ressources système, de décrire des scéna-

rios complexes de montée en charge sur une infrastructure distribuée et de rejouer ces scénarios sur la même infrastructure. Ce constat nous a amené à proposer une nouvelle solution pour l'évaluation des IaaS nommée STRESSCLOUD et présentée ci après.

5.3 STRESSCLOUD : un cadriciel pour l'évaluation des gestionnaires de Clouds

5.3.1 Définitions et architecture

L'objectif de STRESSCLOUD est la définition, l'injection et la synchronisation des activités système d'un groupe de VM au sein d'un IaaS. Cet outil permet de décrire et d'exécuter un scénario d'injection d'activités dans une infrastructure virtualisée distribuée (IaaS). Nous appelons dans cette section *l'administrateur* la personne responsable du déploiement de STRESSCLOUD dans un centre. Nous appelons *l'utilisateur* de STRESSCLOUD la personne en charge de la description des scénarios d'activités et de leur exécution via l'outil STRESSCLOUD. L'outil STRESSCLOUD ne prend pas en charge la création et le déploiement des VM du centre nécessaires à l'exécution d'un scénario. Cette tâche est confiée à l'administrateur, cependant nous mettons à disposition des programmes java et des VM pré-configurées pour mettre en place les différents éléments qui composent STRESSCLOUD.

Dans notre contexte, un *scénario* d'activités est une liste de demande d'activités système, mises en séquence par des contraintes de succession ou de temps, et appliqués à des VM préalablement spécifiées. Par exemple dans cette optique,

```
- vm1 et vm2 sont deux vms du centre
- le CPU de vm1 est à 10% pendant 5 min
- vm1 envoie 10 MB de données à vm2 à partir de la 2 ème minute
- quand vm1 a terminé d'envoyer ses données sur le réseau
   le CPU de vm2 est à 20% jusqu'à la 5ème minute.
```

est un scénario d'activité formulé en langage humain.

Nous nommons *stresseur* le code logiciel chargé de générer une activité dans une machine, par exemple virtuelle, qui l'embarque. Ainsi le stresseur CPU est responsable de l'injection d'une activité CPU dans sa VM. Les stresseurs spécifient chacun leur unité, ainsi l'activité d'un stresseur réseau peut être spécifiée comme un nombre d'octets à émettre sur la carte par seconde sur une connexion UDP, ou bien le nombre de création et fermeture de connexion TCP à effectuer par minute selon l'implémentation du stresseur. De la même manière l'activité d'un stresseur CPU peut être exprimée, comme c'est souvent le cas dans la littérature, en pourcentage du temps de calcul du processeur utilisé, en utilisation de la fréquence processeur ou bien en nombre d'opérations arbitraires à effectuer par seconde. Une VM utilisée pour un test par notre outil

peut donc exécuter plusieurs stresseur, chacun dédié à l'injection d'activité sur une ressource. L'ensemble des activités de ces stresseurs est contrôlé à distance par un système centralisé nommé le *registar*.

Le *registar* est un service d'enregistrement, d'indexation, de sélection et de réservation des VM disponibles pour l'exécution d'un scénario. Il est couplé avec un *exécuteur* de script qui reçoit les scénarios utilisateur et les applique dans STRESSCLOUD. Un registar peut être accessible depuis plusieurs exécuteurs différents afin d'injecter des scénarios en parallèle ou de permettre une plus grande flexibilité d'utilisation.

Dans ce registar la sélection des VM participant à un scénario se fait sur des critères d'environnement virtuel de chaque VM, tel la quantité de mémoire disponible, le nombre de cœurs CPU disponibles ou l'adresse IP de la VM. Le processus de communication entre les stresseurs et le *registar* est confié à une brique logicielle spécifique : l'*exporteur*.

Un *exporteur* est un programme exécuté dans une VM qui établit un canal de communication entre les stresseurs de celle-ci et le registar, typiquement via le réseau. Les exporteurs sont spécifiques à la couche de communication utilisée entre les stresseurs des VM et le registar. Un exporteur ne peut donc se connecter qu'à un registar qui supporte sa couche de communication. Ainsi un exporteur local ne peut se connecter qu'à un registar local, un exporteur tcp ne peut se connecter qu'à un registar intégrant une couche de communication tcp.

Une fois qu'une VM est en communication avec un registar, nous appelons *registeredVM* l'identification de cette VM et de ses stresseurs telle que perçue au sein du registar. Cette abstraction est exposée par le registar à l'exécuteur de script et met à disposition les différentes opérations d'injection d'activité, d'observation de performances, de séquencement nécessaires à l'injection d'un scénario d'activités complexe.

Cette architecture est présentée dans le schéma 5.1.

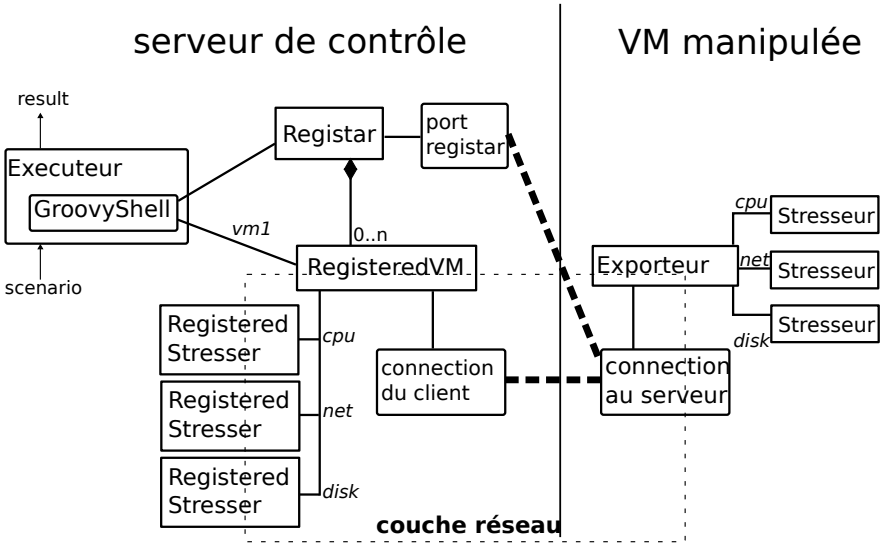


FIGURE 5.1 – Architecture de STRESSCLOUD

La première étape de l'implémentation de cette architecture est la spécification et le développement des stresseurs, puisque leur utilisation conditionne le langage des scénarios.

5.3.2 Génération des activités

L'injection d'une activité système a pour but de produire, dans une VM spécifiée, une activité visible par le SGIV. Cette activité est créée par les stresseurs, qui accèdent de manière contrôlée à une ou plusieurs des ressources disponibles depuis la VM. Selon la littérature, l'activité ressource dans un serveur peut être observée comme

- un nombre d'accès à réaliser à la ressource. Par exemple, lorsqu'un programme résout un problème complexe, il monopolise l'accès à (au moins) un cœur CPU, et exécute un nombre d'opérations fixé dépendant du problème à résoudre.

- une quantité d'accès à effectuer de manière périodique sur la ressource. Par exemple, l'activité d'un serveur streaming en réseau peut être observée comme le débit d'émission de données sur le réseau.

Nous définissons une demande d'activité sur une ressource correspondante, dans le premier cas comme un *travail* à effectuer sur la ressource, dans le deuxième cas comme une *charge* à appliquer à la ressource. Un travail sera donc spécifié comme un nombre total d'accès à effectuer au plus tôt. Ainsi allouer un travail de 1000 à un stresseur de CPU revient à lui demander d'exécuter 1000 instructions processeur au plus vite.

Une charge sera spécifiée comme un taux d'utilisation de la ressource, associé à une durée d'utilisation de cette ressource. Par exemple, allouer une charge de 100/s sur 20 secondes à un stresseur de CPU correspond à une demande d'exécution périodique de 100 instructions processeur toutes les secondes, ceci pendant 20 secondes.

L'évaluation d'une injection d'activités suppose, en plus de la spécification des activités des ressources manipulées, l'observation des activités réelles des stresseurs. En effet, puisque l'objectif de STRESSCLOUD est l'évaluation des gestionnaires, il est indispensable au minimum de comparer l'activité demandée par l'utilisateur avec l'activité réellement exécutée. Cette comparaison, qui peut être vue comme une métrique des performances du SGIV, permet de comparer des gestionnaires, ou d'affiner le paramétrage d'un gestionnaire. Pour permettre cette évaluation les stresseurs mettent à disposition de l'utilisateur des valeurs de performance associées à chaque demande d'activité. Dans le cas d'une demande de charge, la performance est le taux d'activité effectivement exécuté pendant la durée spécifiée. Dans le cas d'un travail, l'utilisateur a accès à la durée d'exécution de celui-ci et à l'avancement du travail au travers du temps.

Pour un même scénario (c'est-à-dire une suite d'activités), l'objectif d'un gestionnaire sera de minimiser les temps de calcul des travaux à effectuer et les taux d'erreurs (rapport entre charge demandée et charge exécutée) des charges à appliquer. Afin que ces comparaisons aient du sens, STRESSCLOUD reproduit de la manière la plus identique possible un même scénario dans ses exécutions successives. Pour cela, le registar assure des propriétés de déterminisme dans la sélection des VM, de même les stresseurs assurent des formes de déterminisme dans l'exécution des activités qui leurs sont demandées.

La section suivante présente l'approche, la méthodologie et les implémentations des différents stresseurs dans STRESSCLOUD.

Unités d'activité des ressources

Si dans la littérature la charge d'une ressource peut être vue comme un taux, sous forme de pourcentage, d'utilisation de la ressource, dans le cadre de la montée en charge d'un système IaaS cette notion de pourcentage n'a pas de sens. En effet, nous souhaitons simuler une succession d'activités système dans un centre où les VM peuvent être déplacées à chaud dans une infrastructure hétérogène. Or, dans un système réel, le taux d'accès à une ressource induit par l'exécution d'une application réelle, en pourcentage d'occupation de cette ressource, varie en fonction des capacités du serveur qui exécute cette application. Dans une infrastructure virtualisée, le serveur hôte d'une VM étant modifiable dynamiquement via les demandes de migrations, les capacités maximales d'accès aux ressources des applications peuvent varier dans le temps. De plus, l'exécution concurrente de plusieurs VM sur un même serveur hôte, de même que les mécanismes de gestion des ressources dans un SGIV, peuvent réduire la capacité d'accès à une ressource perçue par chaque VM.

C'est pourquoi les accès aux ressources des VM doivent être spécifiés dans une unité qui ne dépende pas des capacités du serveur hôte. Ainsi, la charge usuelle de la ressource CPU, sous forme de pourcentage d'utilisation du temps CPU disponible, n'a plus de sens dans un scénario de montée en charge virtualisé. Nos stresseurs utilisent des unités telles que les MHz d'un CPU ou le débit d'écriture (en MB/s) d'un disque, voire des unités propres au stresseur développé.

Si le mécanisme de génération d'une activité est spécifique à la ressource à manipuler, l'architecture et les principes de fonctionnement des stresseurs sont identiques et présentés ci-après.

Modèle générique de stresseur périodique

Les stresseurs proposés dans STRESSCLOUD utilisent tous le même schéma d'injection d'activité présenté en 5.2. Un stresseur générique effectue de manière périodique des accès sur la ressource qu'il manipule. L'exécution effective des accès à la ressource dépend du stresseur développé, il est donc nécessaire de développer un stresseur pour chaque type d'accès à une ressource.

Lorsque l'activité demandée par l'utilisateur est une charge, la période des cycles d'injection, aussi appelée granularité de l'injection, est configurable par l'utilisateur. Dans ce cas, le nombre d'accès à effectuer à chaque cycle est le produit de la période des cycles avec la fréquence d'activité demandée par l'utilisateur. Si l'utilisateur demande 50 accès par seconde avec une granularité de 10ms, alors le stresseur effectuera 5 accès à chaque cycle.

Ce stresseur générique permet d'observer le taux d'erreurs d'une demande de charge : à chaque début de cycle, le stresseur calcule si la durée d'exécution du cycle précédent a dépassé la période de cycle. Dans ce cas, le stresseur ajoute cette durée de dépassement de cycle à une somme interne. D'un instant à l'autre, le quotient de la différence de cette somme par la différence de temps entre ces deux instants donne le taux d'erreurs de l'activité demandée, c'est à dire le taux d'activité qui n'a pas pu être effectué.

Le paramétrage de la granularité d'un stresseur est une tâche délicate. En effet, plus la granularité est basse (c'est-à-dire la période de cycle est courte), plus les fonctions de gestion du stresseur sont appelées fréquement. De plus, une période proche de celle de la politique d'ordonnancement du système peut induire des taux d'erreurs importants. Ainsi, si un système ordonne les processus sur des périodes de 1 ms, nous ne pouvons pas demander une granularité inférieure à cette valeur à un stresseur.

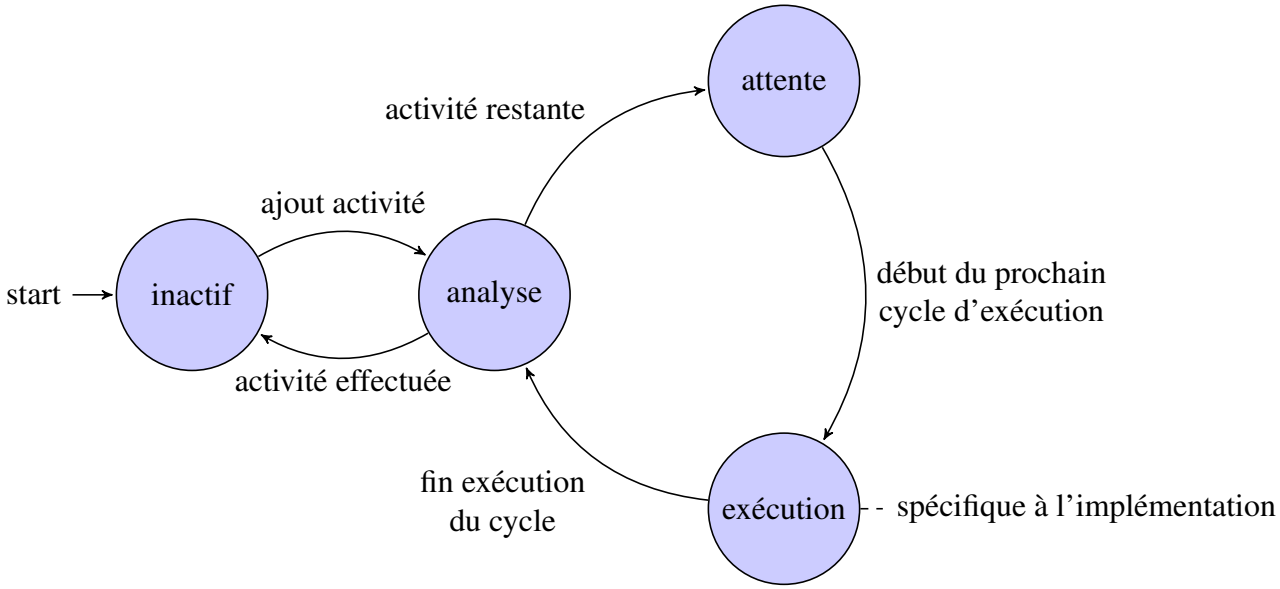


FIGURE 5.2 – Activité d'un stresser générique périodique.

Lorsque l'activité demandée par l'utilisateur est un travail, le stresseur se limite à exécuter l'ensemble des instructions le plus rapidement possible et mémorise le temps qui a été nécessaire pour exécuter ces instructions.

Les sections suivantes présentent les principes et implémentations des stresseurs proposés par STRESSCLOUD.

stresseur CPU

Le taux d'utilisation du CPU est souvent utilisé pour représenter l'activité d'une VM dans un IaaS. L'outil le plus communément utilisé pour charger le CPU d'une machine est, sous linux, *stress*. Cet outil permet de stresser au maximum le CPU, la mémoire ou le disque de la machine sur laquelle il est exécuté. D'autres outils sont disponibles comme *cpuburn* ou *stressapptest*, malheureusement ces outils se limitent à tester la capacité maximale d'un processeur, et ne permettent donc pas de stresser un processeur à une charge ou un travail donné.

Les accès CPU sont généralement observés depuis un serveur sous la forme de la fraction de temps processeur utilisée pour l'exécution des programmes de ce serveur. Cependant, ces observations n'ont plus de sens dans une infrastructure virtualisée où les capacités de calcul des processeurs sont variables.

L'activité CPU d'une VM peut aussi être visualisée dans un serveur par la fréquence des cycles d'horloges du processeur qui sont attribués à l'exécution de cette VM durant une seconde, en Hz. Malheureusement, l'injection d'une activité en Hz dans une

VM rencontre aussi des problèmes liés aux différents modèles de processeurs installés. En effet, cette injection doit en effet réalisée par un code commun aux différents serveurs. Cependant, les différentes architectures de processeurs nécessitent des nombres de cycles différents pour réaliser la même opération.

Nous avons comparé la consommation, en tick d'horloge CPU, de différents modèles de serveurs pour exécuter la même tâche. Cette tâche consiste en l'exécution d'un nombre fixé d'opérations de type `sqrt`, exécution répétée plusieurs fois et dont le temps d'exécution a été mesuré au moyen de l'instruction processeur `rdtsc`. Nous avons retenu le temps d'exécution de la tâche le plus faible parmi 500 évaluations, en demandant pour chaque évaluation 2048 exécutions de l'instruction `rdtsc`. Nous nous sommes assurés que pour des nombres moins importants d'itérations les différences entre les temps d'exécution étaient du même ordre.

Nos serveurs de test sont un Apple MacBook Pro, un Dell Studio Hybrid, un HP Z200 et un Dell R415. Le résultat de ces évaluations, indiqué sur la figure 5.3, montre que le nombre de cycles requis pour exécuter la même tâche varie du simple au double selon les serveurs.

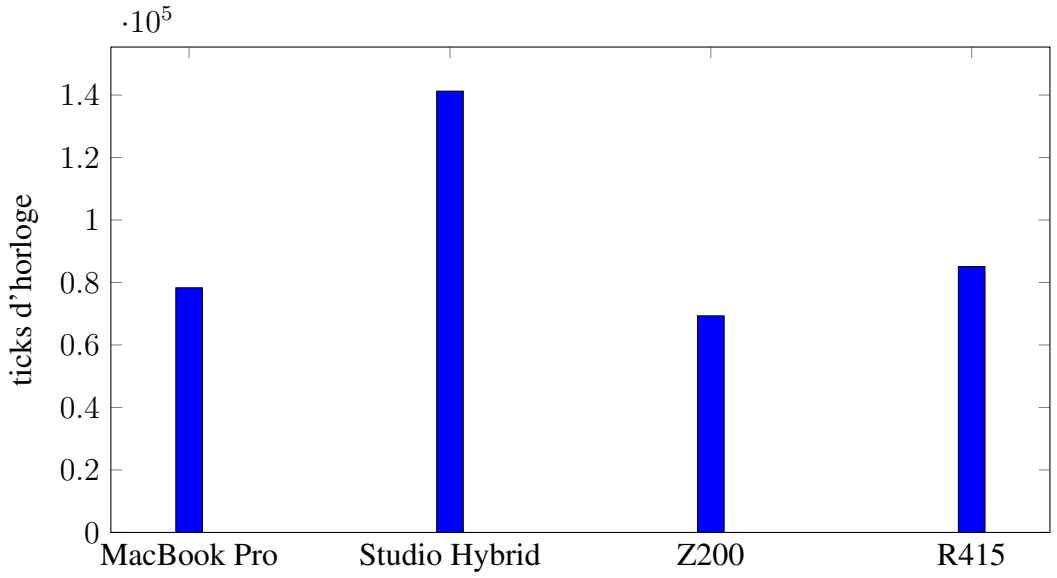


FIGURE 5.3 – ticks d'horloge pour 2048 opérations sqrt

Avec une telle marge d'erreur il n'est pas possible d'assurer la fréquence d'utilisation du CPU indépendamment de son architecture. De fait, l'activité CPU que l'on observe sur un serveur est fonction de la séquence d'instructions à exécuter par le processeur autant que de l'architecture de celui ci, et ne permet pas de produire un programme qui reproduise la même activité sur tous les serveurs. Les processeurs récents permettent de mesurer plus finement l'activité du CPU, en implémentant un système de

compteur physique des performances des programmes. Ces compteurs de performances consistent en des registres dédiés chacun à l'enregistrement d'un événement CPU. Ainsi le nombre d'instructions exécutées, de défauts de page mémoire, de sauts conditionnels sont enregistrés pour chaque programme. Ces compteurs de performances ne sont cependant pas utilisés dans les centres virtualisés à notre connaissance.

Puisqu'il est complexe de charger un processeur à une fréquence donnée, mais surtout de corréler les fréquences processeur lors de la migration d'une VM sur un autre modèle de serveur, nous avons opté dans notre implémentation pour une unité propre au stresseur, un nombre de boucles à exécuter chaque seconde. En effet, il est a remarquer qu'une application ne consomme pas de MHz ni de bogoMIPS, mais bien des instructions processeur. Ces instructions sont généralement très variées et il est donc plus parlant pour l'utilisateur de STRESSCLOUD de charger un processeur suivant un nombre d'instruction a exécuter.

Cette unité n'assure pas de valeur de fréquence d'accès au CPU dans des unités usuelles (MHz, bogoMIPS, temps d'utilisation CPU). Il est cependant possible de calibrer cette unité sur un serveur pour en déduire une relation entre la demande d'activité du stresseur et la charge CPU à injecter dans ce serveur.

stresseurs d'activités disque et réseau

Si l'injection d'une activité CPU contrôlée dans un serveur est une tâche délicate, les injections d'activité réseau et disque sont par contre beaucoup plus simples à condition de limiter les fonctionnalités. Ainsi les performances de la lecture sur un disque sont grandement impactées par les phénomènes de cache, tandis que la lecture d'un flux de données du réseau suppose la présence d'un émetteur de ce flux. Si des techniques permettent de passer outre le cache disque ou d'envoyer des requêtes réseau vers soi-même, elles demandent une connaissance du système qui reste erronée dans le cadre d'un environnement virtualisé, où le système virtualisé est susceptible d'évoluer suite à des migrations ou des changements de paramétrage du serveur. C'est pourquoi nous nous limitons pour ces deux ressources à des accès en écriture.

Ces deux stresseurs implémentent une activité d'envoi de données, pour le disque vers un fichier, pour le réseau vers une adresse cible. Dans les deux cas l'envoi doit avoir une destination valable, sinon l'activité d'écriture demandée sera annulée, par le système d'exploitation ou l'hyperviseur du serveur par exemple. Enfin, le processus d'envoi de données doit utiliser le protocole du plus bas niveau possible pour éviter la possibilité pour le système d'exploitation de la VM d'optimiser, et donc réduire, cette activité.

Le stresseur réseau est simple à implémenter, car toute donnée écrite sur la carte réseau doit être envoyée le plus tôt possible. Ce stresseur est paramétré par une adresse réseau, spécifiée au besoin par l'utilisateur, à laquelle il se connecte et envoie des paquets de données par le protocole UDP. Ce protocole UDP a des propriétés intéressantes

pour la simulation d'activité : il ne nécessite pas de connexion réelle avec un autre serveur, n'utilise pas de système de tampon, et effectue le minimum de vérifications sur l'envoi effectif, et la réception effective, des paquets par le serveur. De plus, le protocole UDP ne fait pas de compression de données, ce faisant l'envoi d'un suite de N bits valant chacun "0" sur le réseau enverra effectivement ces N bits.

L'adresse d'envoi par défaut est l'adresse du réseau (le netID) utilisé pour se connecter au registar. Cette connexion étant nécessaire dans STRESSCLOUD, cette adresse est connue par l'exporter et le registar. Par exemple, si la carte de réseau a l'adresse 192.168.0.1/24, le réseau associé sera 192.168.0.0. En cas de registar local ou d'utilisation d'un protocole de communication différent, des méthodes de récupération automatique de l'adresse IP du réseau sont utilisées, mais l'administrateur est responsable de la spécification, lors de l'installation des VM, de l'adresse IP utilisée pour les atteindre si celle-ci n'est pas évidente. L'adresse du réseau local est intéressante car elle ne correspond à aucune carte réseau, ainsi les paquets qui y seront envoyés seront bien émis par la carte réseau mais détruits par le premier routeur du réseau, qui ne saura pas vers qui les transmettre.

Deux autres adresses génériques pourraient être utilisées : l'adresse de diffusion réseau, dans l'exemple précédent 192.168.0.255, ou l'adresse de diffusion limité, 255.255.255.255. Cependant ces deux adresses ont le défaut de demander la diffusion effective des paquets sur tout le réseau accessible. Les messages seraient donc lus et éventuellement analysés par l'ensemble des machines présentes sur le réseau, créant donc un effet de bord non désirable et réduisant la bande passante disponible sur le réseau. De plus, l'utilisation de tels procédés peut induire des mécanismes de sécurité à limiter l'accès de la machine émettrice au réseau, réduisant par la suite la bande passante disponible pour cette dernière, voire restreignant son accès au réseau, faisant ainsi apparaître des pannes inexpliquées.

Enfin, l'utilisateur peut spécifier cette adresse d'envoi de paquets, permettant par exemple l'envoi de données d'une VM à une autre. Cette technique permet de simuler un canal de discussion entre deux VM, canal qui peut parfois être observé selon le SGIV utilisé. Cependant chaque VM ne peut actuellement envoyer des données que vers une seule autre VM. La prise en compte de l'émission multi-cible est une évolution prévue de l'architecture de STRESSCLOUD mais nécessite quelques travaux de langage avant d'améliorer le stresseur réseau.

Le stresseur de disque quant à lui doit prendre en compte les détails d'une implémentation du système d'écriture sur disque. Premièrement, il est nécessaire d'utiliser un protocole qui assure l'écriture sur le disque des données émises par la VM. En effet la création de fichiers temporaires depuis un programme peut être traduite par l'OS par la création d'un fichier en mémoire, qui ne sera donc pas écrit sur le disque. Pour cette raison, nous utilisons des fichiers réels sur lesquels nous écrivons les données, au contraire de l'activité réseau qui peut se passer d'une cible réelle atteignable.

D'autre part, la capacité des disques étant limité, nous devons limiter la taille des données écrites par ce stresseur. Bien que les fonctionnalité de suppression du fichier et de création d'un nouveau fichier soient facilement accessible, ces fonctionnalité ont un impact non négligeable sur la charge CPU du système. Nous utilisons la fonction système `seek`, qui permet de réutiliser le même fichier déjà présent sur le disque et nous assure une surcharge CPU négligeable. Le fichier recevant les données écrites est supprimé automatiquement par la JVM lors de la terminaison de celle-ci, et une demande de suppression au redémarrage du système est ajoutée pour éviter que l'arrêt abrupte de cette JVM ne puisse produire une fuite d'espace disque.

Les mécanismes de tampon disque présents sur un OS peuvent court-circuiter l'écriture des fichiers, par exemple en maintenant les données à écrire en mémoire. Ainsi lors de la suppression des données écrites du disque, de tels mécanismes peuvent effacer les données en mémoire sans que ces dernières aient été effectivement écrites sur le disque. C'est pourquoi il faut s'assurer dans le protocole utilisé que les données sont effectivement écrites sur le disque avant d'effacer ces dernières, par exemple avec les fonctionnalités de synchronisation du système de fichier, *eg.* la commande `sync` sous linux et ses dérivés.

Enfin, la gestion des systèmes de fichiers étant complexe, il faut s'assurer d'avoir les droits de création, d'écriture et de suppression des fichiers depuis le programme du stresseur. Ces contraintes sont à la charge de l'administrateur qui doit installer STRESSCLOUD dans un environnement adéquat. Nous rappelons que des VM pré configurées sont mises à disposition dans STRESSCLOUD

Notre implémentation du stresseur de disque utilise la classe d'écriture de flux Java `FileOutputStream` pour ajouter des données à un fichier. Cette classe possède des propriété qui nous semblent suffisantes pour cette utilisation. D'une part, le tampon mémoire du flux peut être vidé afin d'assurer l'écriture effective des données qu'il contient, permettant de se déplacer dans le fichier sans annuler les demandes d'écriture précédentes. D'autre part, cette structure n'utilise pas de système de compression des données, permettant d'écrire plusieurs fois de suite la même structure en mémoire sans craindre que l'homogénéité des données n'induise une réduction de l'activité effective de la ressource.

Manipuler les stresseurs via des API de programmation serait une tâche ardue. En effet, chaque VM peut potentiellement héberger plusieurs stresseurs, et il est nécessaire de disposer de plusieurs dizaines voir centaine de VM pour évaluer des gestionnaires d'infrastructure. De plus, en vue de simuler le fonctionnement d'application, il est indispensable de pouvoir synchroniser les activités entre elles De ce fait, nous nous sommes orientés vers la définition d'un langage dédié au contrôle des activités. Ce langage, interprété par un interpréteur dédié, doit permettre d'exprimer aisément des charges ou travaux sur les activités, mais également d'exprimer des contraintes de précédences entre activités.

Les fonctionnalités offertes par ces implémentations de stresseurs nous ont permis de déduire celles à intégrer dans le langage. En pratique, la description d'un scénario utilisateur se limite généralement aux fonctionnalités communes parmi les stresseurs. C'est pourquoi ce langage permet d'utiliser facilement ces fonctionnalités commune, tout en permettant à un développeur averti de configurer dynamiquement, via ce langage, les paramètres des implémentations réalisées. Ainsi, un utilisateur peut décrire des scénarios simples, tandis qu'un développeur peut évaluer l'impact des paramètres d'un stresseur via un scénario plus complexe.

La section suivante décrit ce langage.

5.3.3 Un langage dédié pour le contrôle des activités

Nous avons développé un langage de script permettant de décrire des scénarios et un interpréteur de ce langage pour exécuter ces scénarios. Ce langage permet de sélectionner les VM à manipuler parmi celles enregistrées sur le registar, puis d'injecter des activités ordonnancées dans celles-ci. Ce langage est suffisamment simple pour être utilisé par un non-développeur, mais il permet aux développeurs de développer des extensions, par exemple en REST, pour manipuler ou observer les VM du centre.

Afin d'éviter les écueils du développement d'un langage spécifique, nous nous sommes appuyés sur le langage Groovy [BS06], définissant un sur-langage de celui ci. En particulier nous utilisons la classe *Groovyshell* pour interpréter et exécuter les scripts utilisateur. L'utilisation de Groovy a plusieurs avantages sur Java :

- Le langage Groovy simplifie grandement la manipulation des collections, ce qui est utile pour la manipulation en masse de centaines de VM
- L'environnement GroovyShell implémente l'interprétation et l'exécution de scripts Groovydans une JVM standard
- Cet environnement propose la gestion de variables, l'invocation des méthodes et l'accès à des objets internes
- Notre langage bénéficie ainsi de l'ensemble des ces fonctionnalités sans aucun développement, nous nous concentrons sur les fonctionnalités propres à notre domaine

Nous distinguons trois phases dans le développement de scénario. La première phase d'un scénario est chargée d'identifier et de sélectionner les VM nécessaires à l'exécution du scénario. La seconde phase spécifie les charges et travaux à exécuter au sein des VM, ainsi que la synchronisation entre ces charges. Enfin, la troisième phase d'un scénario permet d'obtenir des mesures de performance de l'exécution du scénario. Les sections suivantes présentent les mécanismes et abstraction liés à ces trois phases.

Sélection des VM

La phase de création et de paramétrage des VM n'est pas prise en charge par STRESSCLOUD. En effet, cette partie est très spécifique au système de IaaS utilisé. L'administrateur doit donc déployer les applicatifs de STRESSCLOUD sur le IaaS à manipuler, puis configurer (nombre de VCPU, taille mémoire etc.) et démarrer l'ensemble des VM nécessaires au scénario à exécuter. Toutes les VM démarrées s'enregistrent sur le REGISTAR. Ce dernier centralise ainsi l'ensemble des caractéristiques des VM présentes (VCPU, RAM, adresse ip etc.).

La première étape de l'écriture du scénario consiste à sélectionner et identifier un ensemble de VM nécessaires à l'exécution du scénario. Cette sélection permet de créer des groupes de VM, mais aussi de comprendre et modifier plus facilement un scénario. Les critères de sélection considèrent les paramètres des VM connus du REGISTAR, et donc fournis par les VM.

Cette sélection doit être déterministe, afin que l'exécution du même script dans les même conditions produise la même sélection de VM. Étant donné que les VM d'un centre peuvent migrer dans celui-ci, l'exécution du même scénario peu cependant produire des activités différentes. L'administrateur doit donc, entre deux exécutions du même scénario, migrer les VM sur leurs serveurs initiaux, ce qui ne peut être automatisé dans STRESSCLOUD. Si cette fonctionnalité de re-placement n'est pas disponible, par exemple lors de l'évaluation des performances d'un IaaS externe comme AWS, l'utilisateur peut toutefois lancer plusieurs fois le même scénario et faire une moyenne de performances.

La méthode `group=require(critère, nombre)` réserve un `nombre` de VM respectant les caractéristiques `critère` données en paramètre. Le critère peut être `null`, dans ce cas les premières VM disponibles seront sélectionnées pour réservation. Ce critère peut porter sur le nombre de cœurs CPU alloués, la taille de la RAM, et l'adresse IP de cette VM. Par exemple, le code `group3=require("mem>5000&ip== '192.168.3..` `25)` réserve 25 VM de plus de 5 GB de RAM et dont l'adresse IP correspond à 192.168.3.X
.

Cette méthode retourne l'ensemble (`set`) de VM qui ont été réservées par cette méthode. Si le nombre de VM disponibles sur le registar n'est pas suffisant, cet appel est suspendu pour permettre à d'éventuelles VM non connectées de terminer leur enregistrement. Pour éviter une exécution trop longue du script de sélection, un *timeout* est configurable à la fin duquel, si le système n'est pas parvenu à sélectionner le nombre de VM nécessaires, la méthode renvoie `null` et ne réserve aucune VM.

Après avoir utilisé les VM pour exécuter du scénario, la méthode `release(group)` libère les VM réservées dans `group` et met leurs activités à 0. Un appel à cette méthode sans paramètre libère l'ensemble des VM réservées du registar, permettant l'exécution d'un nouveau scénario.

Un des intérêts de GroovyShell est la facilité de manipulation des collections. De

telles opérations permettent de manipuler aisément des groupes de VM, que ce soit pour la création de sous-groupes ou l'injection d'une même activité dans un groupe de VM. Par exemple l'instruction `g1=group[0..9];g2=group[10..99]` définit g1 comme les dix premiers éléments de `group` et g2 comme les 90 suivants. Les opérations de manipulation d'activité sont décrites dans la section suivante.

Manipulation des activités des ressources

Une fois les VM sélectionnées, des méthodes dédiées permettent de manipuler leurs activités système. Ces méthodes sont générique et permettent une intégration aisée de nouveaux stresseurs de ressource dans le langage. Aussi, seules les noms des ressources manipulables dans une VM sont connues du registar. Les implémentations effectives de leurs stresseurs ne sont donc pas connues du registar, mais seulement depuis les VM.

Lorsqu'elle s'enregistre dans le registar, une VM manipulable lui spécifie quels sont les ressources manipulables. Le registar met en place pour chaque VM enregistrée un système de délégation des appels, visible dans l'exécuteur de script, qui traduit les appels de l'utilisateur en commande pour la VM. La méthode `vm.getTypes()` permet de lister les ressources manipulables au sein d'une VM, et donc les différents stresseurs implémentée dans cette VM. Ces stresseurs permettent l'injection d'activité sous deux formes : l'injection d'une charge régulière ou l'ajout d'un travail à effectuer.

L'opérateur d'affectation de charge dans notre langage est `vm1.res=load`. Cet appel positionne l'utilisation de la ressource `res` de la VM `vm1`, et donc l'activité du stresseur correspondant, à une charge `load`/s. Comme précisé plus haut, la sémantique de cette charge dépend de l'implémentation du stresseur. Ainsi, le stresseur CPU peut très bien définir sa charge relativement au temps de calcul disponible, ou en nombre fixé d'opérations à exécuter.

Pour bien différencier la spécification de charge de l'ajout d'un travail, nous avons utilisé un autre opérateur pour cette deuxième opération. Ainsi `vm1.res+work` ajoute une quantité de travail `work` à la ressource `res` de la VM `vm1`. L'exécution de ce travail commence dès que le stresseur correspondant à cette ressource a terminé les éventuels travaux précédents. Ainsi, l'observation des activités des VM permet de rendre compte du démarrage et de l'arrêt effectif des travaux demandés. Ces mécanismes sont génériques quelque soit la ressource manipulée par le stresseur.

Le stresseur réseau fonctionne sur ce principe, mais nous avons ressenti le besoin d'exprimer l'émission de données d'une VM vers une autre. En effet, les communications réseau des VM sont une activité importante dans un centre. Nous avons développé une implémentation spécifique dans le langage, pour réduire la taille des scénarios et donc le travail de l'utilisateur. Ainsi, la méthode `vm1.send(vm2, size)` produit l'envoi par la VM `vm1` via son stresseur de type `net` d'une quantité `size` de données vers l'adresse IP de la VM `vm2`.

Cette méthode est un raccourci vers des opérations internes sur les stresseurs. Plus

précisément cette méthode utilise la sélection d'une cible du stresseur, puis l'ajout de travail dans ce stresseur. Or la sélection de la cible d'un stresseur n'est pas toujours possible, par exemple dans le cas d'un stresseur CPU. Cette méthode suppose donc que le stresseur de ressources réseau de la VM supporte la fonctionnalité de sélection de cible.

Outre ces ajouts de charges et travaux dans le centre, nos scénarios comportent des systèmes de synchronisation de ces activités. Cette fonctionnalité de synchronisation des activités est décrite ci-après.

Planifaction d'un scénario

La planification d'un scénario d'activité consiste à délayer l'ajout des activités spécifiées. Après étude et évaluation d'un certain nombre de modèles applicatifs, nous avons déterminé que la totalité des scénarios que nous souhaitons décrire peuvent se limiter à deux méthodes de synchronisation. La première méthode démarre une activité au bout d'un certain temps T. La deuxième méthode attend la fin d'une (ou plusieurs) autre(s) activité(s).

En plus de ces fonctionnalités, l'outil permet la synchronisation intra-scénarios. Ce terme signifie que l'ensemble de ses instructions peut être synchronisé, de manière homogène avec la synchronisation des VM. Ainsi, les méthodes de synchronisation inter-VM, qui mettent en relation des changements d'activité avec des événements, sont applicables à un scénario. Ils bloquent donc l'ensemble des instructions du scénario en attendant un événement.

La première méthode de synchronisation est la synchronisation temporelle, elle peut être exprimée de deux manières. L'utilisateur peut spécifier une attente durant un délai absolu, c'est-à-dire durant un intervalle de temps donné. Par exemple, un scénario peut charger un CPU à 50 pendant 2min.

Cette synchronisation temporelle peut aussi être spécifiée par rapport au début du scénario, par exemple jusqu'à la 3^{me} heure. Pour cela, l'exécuteur, responsable de l'exécution du script, doit avoir connaissance du début de l'exécution d'un scénario. En pratique le début du script est défini par l'heure de la dernière exécution d'une commande `require` par le scénario. Ainsi, un scénario commence lorsque sa dernière instruction `require` retourne son résultat, il se termine dès qu'un appel à `release` est effectué.

Le séquencement temporel absolu d'activités est exprimé par la méthode `vm.after(seconds)`. Cette commande demande à la VM d'attendre `seconds` secondes avant d'envoyer les commandes suivantes. L'expression du séquencement relatif est implémenté par la méthode `vm.till(seconds)`. Cette dernière demande à la VM la suspension des commandes d'activité jusqu'à ce que le script ait atteint ses `seconds` secondes d'exécution.

Pour faciliter la description de scénarios simples, la méthode `lb=loadBase(time, load)` crée une spécification de charges partagées `lb`. Cette charge partagée spécifie une période d'exécution des charges, ainsi qu'une valeur de charge de base. Elle est

appliquée à différents stresseurs pour mutualiser des demandes d'activité, réduisant la verbosité des scénarios. Par exemple, `lb.to(vm1.cpu, mult1, mult2)` spécifie la charge du stresseur `stresser` à $mult1 \times load$ pendant `time` secondes, puis à $mult2 \times load$ pendant la même durée, pour finalement mettre la charge à 0. Cette méthode permet de monter en charge toutes les VM d'un centre en même temps à la même activité, par exemple pour calibrer des modèles de consommation électrique des serveurs et du centre.

Le séquencement temporel des activités peut être effectué au niveau du scénario. Ceci permet par exemple d'effectuer en séquence plusieurs scénarios, après des temps de repos ou d'ajustement du centre. Ainsi `after(seconds)` délaye l'interprétation du script du temps demandé, tandis que `till(seconds)` la délaye le temps que le délai indiqué se soit écoulé depuis le début de l'exécution du scénario.

Durant cette suspension, les VM qui ont des activités en attente continuent cependant d'envoyer leurs commandes aux stresseurs dès que leurs conditions d'attente sont terminées. Par exemple, le script suivant fait attendre le scénario 15 secondes, durant lesquelles l'attente de la VM de deux secondes sera terminée. Celle-ci aura donc modifié l'activité CPU en la mettant à 200 durant cette suspension.

```
vm.cpu=100
vm.after(2).cpu=200
after(15)
vm.cpu=0
```

La synchronisation des travaux des VM est quant à elle implémentée par l'abstraction `vm.after(group)`. Cette appel empêche l'envoi des commandes d'activité de la VM jusqu'à ce que tous les stresseurs et VM de `group` aient terminé tous les travaux qui leur ont été demandés avant l'appel à `after`. Cette méthode est aussi disponible sous la forme raccourcie `vm.stresser.after()` qui est un équivalent à `vm.after(vm.stresser)`.

De même que pour le séquencement temporel, le séquencement des travaux est présent au niveau du scénario. Ainsi `after(group)` suspend l'interprétation du script tant que des travaux dans les stresseurs et VM de `group` ne sont pas terminés. Ceci permet la création de points de synchronisation dans un scénario, par exemple à la fin de celui ci. Il peut en effet s'avérer dommageable de libérer les VM du centre avant que celles-ci aient effectivement terminé les travaux demandés. Ainsi la commande `after(reservedVMs()); release()` assure que toutes les VM réservées par le scénario ont terminé leurs travaux avant de terminer le scénario.

Une fois nos VM spécifiées, nos activités décrites et planifiées, STRESSCLOUD nous permet d'observer les performances du scénario exécuté.

Observations et analyse d'une exécution

L'objectif de STRESSCLOUD est de permettre d'évaluer les performances d'un SGIV. Pour cela, il faut donc permettre de définir et exécuter un scénario, mais aussi observer l'exécution effective de ce dernier, c'est-à-dire les performances obtenues lors de l'exécution des activités.

STRESSCLOUD met à disposition de l'utilisateur des statistiques d'exécution des activités, basées sur une définition homogène du temps durant l'exécution du scénario. Ces statistiques contiennent la liste des travaux et charges demandés dans le scénario, ainsi que leurs dates de démarrage et de terminaison. Elles contiennent aussi les relevés de performances associées à chaque demande d'activité.

La surveillance d'un scénario est démarrée dès que celui-ci commence, c'est-à-dire dès le premier appel à la méthode `require`. Le temps de départ du scénario est celui du dernier retour de cette méthode, aussi cette surveillance ne considère que les événements ayant eu lieu depuis ce dernier retour. Elle s'arrête dès que l'utilisateur appelle une des méthodes `release`. En effet ces méthodes permettant l'utilisation des VM dans d'autres scénarios, les données de surveillances ne peuvent plus assurer de cohérence avec le scénario actuel, et sont alors supprimées. Durant le scénario, ce système enregistre des demandes de charges et de travaux envoyées aux stresseurs des VM, ainsi que leurs temps d'exécution effectifs et leurs performances associées.

Les valeurs de performance sont obtenues automatiquement à chaque fois que la charge ou le travail d'un stresseur est modifié. Il est de plus possible de spécifier les intervalles d'observation automatique des performances avec la méthode `vm.ping(seconds)`. Par exemple, l'instruction `vm1.ping(120)` spécifie que les valeurs de performance doivent être demandées au maximum toutes les deux minutes pour la VM `vm1`.

Dès qu'un scénario est démarré, l'utilisateur peut accéder à la liste des activités injectées dans les VM manipulées, qu'elles l'aient été sous forme de travail ou sous forme de charge, ainsi qu'aux valeurs de performances observées.

La méthode `works(workers)` retourne l'ensemble des travaux demandés aux VM et stresseurs de `workers`. Cet ensemble retourné est une liste des travaux demandés. Chaque élément de cette liste contient le temps de démarrage du travail, le temps de fin pour un travail terminé, les observations d'avancement du travail et la quantité initiale de travail demandée.

La méthode d'obtention des statistiques des charges est `loads(loaders)`. Elle retourne la liste des demandes de charge des VM et stresseurs de `loaders`. En plus de la valeur de charge demandée, du temps de démarrage et de fin de la charge, ces demandes de charge contiennent les informations de taux d'erreur. Ces informations de taux d'erreurs sont obtenues automatiquement à chaque fois que la VM termine une instruction de suspension temporelle `vm.after(temps)`, et de manière régulière avec `vm.ping(seconds)`. De plus l'utilisateur peut forcer l'obtention de ces informations par la commande `vm.stresser.skipped` d'un `stresser`, que nous décri-

vons plus bas.

L'ensemble des statistiques peut être exporté au format CSV avec la méthode `toCSV()`. Pour les activités sous forme de travail, `works().toCSV()`, *eg.* `works(vm1.cpu, vm2).toCSV()` affiche au format CSV tous les travaux demandés au stresseur de CPU de la VM`vm1` ainsi que tous les travaux demandés aux différents stresseurs enregistrés par `vm2`. Pour les activités sous forme de charge, les demandes de charge et leurs taux d'erreurs associés peuvent être affichée avec la méthode `loads().toCSV()` . L'exportation des statistiques permet une analyse fine post-scénario.

Si l'exécution d'un scénario se passe de manière non attendue, par exemple suite à une erreur de frappe lors de la saisie du scénario, les méthodes `release` vues plus haut annulent toutes les demandes de charge ou travail parmi des VM et les rendent disponibles à la réservation. En cas de problème plus localisé, des méthodes permettent d'observer et de préempter l'exécution du scénario.

La méthode `vm.res.skipped` retourne le temps d'erreurs pour le stresseur de la ressource `res`. Ce temps d'erreur correspond au nombre de ms d'activité qui auraient été nécessaires pour respecter la charge demandée dans les cycles impartis. Par exemple si cette charge demandée est le double de la charge maximale possible pour la ressource, alors le temps d'erreur sera le même que le temps écoulé depuis le début de la demande de charge. Cette valeur permet d'avoir une idée de la surcharge de la VM pour une ressource donnée.

Concernant les demandes de travaux, la méthode `vm.res.work` retourne le nombre d'activités restantes à exécuter par le stresseur de ressource `res`. Afin de prévenir le blocage de l'utilisateur pour cause d'un travail trop long à réaliser, la méthode `vm.clear()` annule tous les ordres de travaux et charges en attente sur cette VM et arrête toute activité sur ses stresseurs enregistrés.

Ces méthodes permettent d'observer et modifier un scénario en cours d'exécution, en obtenant les valeurs de performances et arrêtant les comportements inadaptés, par exemple une erreur de frappe ou la fin anticipée d'un scénario sans pour autant annuler ce dernier. Elles permettent aussi d'évaluer la correction des implémentations de stresseurs de ressources et de calibrer leurs capacités, par exemple pour atteindre progressivement la charge maximale d'un stresseur et observer les effets de seuils dans le comportement du SGIV responsable.

La section suivante présente une évaluation de STRESSCLOUD, tant sur ces fonctionnalités, son langage que sur ses performances.

5.4 Évaluation

Nous évaluons STRESSCLOUD sous deux perspectives : nous évaluons d'abord les stresseurs, puis la bonne exécution d'un scénario.

L'évaluation des stresseurs vérifie que ceux ci injectent une activité système en relation avec la charge ou le travail demandé. Nous nous assurons aussi que les observations de performances des stresseurs concordent avec les valeurs théoriques. Ces évaluations sont réalisées en injectant des activités successives.

Afin de déterminer la plage de travail des stresseurs, nous les calibrons en leur demandant d'effectuer un travail de taille arbitraire et en observant le temps nécessaire à l'accomplissement de ce travail. La charge maximale calibrée est ensuite définie comme le rapport de la quantité d'activité demandée par le temps d'accomplissement de cette activité. Dans le cas où le temps d'accomplissement de ce travail est inférieur à 2s(valeur arbitraire), nous doublons la quantité de travail à accomplir et relançons la calibration.

En plus d'évaluer l'injection d'activité, nous nous intéressons à la déduction des modèles énergétiques des serveurs. Pour cela, nous utilisons les stresseurs pour produire des données de consommation électrique d'un serveur. En particulier, nous nous intéressons à la répartition de cette consommation parmi les ressources du serveur.

Enfin, nous décrivons deux scénarios pour évaluer l'exécuteur, et nous les injectons dans un centre réel pour observer l'exécution de ce scénario dans le SGIV. Le premier scénario de type HPC permet d'évaluer le bon séquencement de travaux tandis que le deuxième de type webserver permet d'évaluer le séquencement temporel des charges partielles dans le centre. Le scénario de type HPC est exécuté sur un petit centre après une phase de calibrage nous assurant que les valeurs de travaux demandées sont observables par le SGIV. En effet, si ces valeurs sont trop petites les travaux correspondants peuvent être terminés sans que le SGIV n'ait remarqué une quelconque influence sur les activités des VM.

5.4.1 Évaluation des stresseurs

Un stresseur a pour objectif d'accéder de manière contrôlée et périodique à une ressource de sa VM. Nous devons alors nous assurer qu'une demande d'activité injectée dans ce stresseur est visible en observant l'activité effective de sa ressource. Certaines ressources pouvant induire une activité CPU supplémentaire, nous observons la charge CPU induite par l'utilisation des stresseurs manipulés. Nous évaluons la correction des stresseurs de CPU, de réseau et de disque, ainsi que leur surcharge CPU induite.

stresseur de CPU

Dans cette évaluation nous étudions un stresseur de CPU sur un serveur physique, non enregistré sur un registar, pour lequel nous faisons varier la valeur de charge demandée. Nous observons régulièrement la valeur de charge en pourcentage du CPU du serveur, grâce à l'outil `ThreadMXBean` intégré à la plateforme java.

Cette évaluation est effectuée sur un serveur HP Z200 à 4 cœurs CPU, en demandant au stresseur de CPU d'utiliser 2 puis 4 threads, avec une granularité de 200ms.

Chaque modification de l'activité du stresseur est suivie d'une période d'attente de 5s pour permettre au système d'atteindre une stabilité de charge, puis d'observation du taux d'erreurs et de la charge CPU de la JVM pendant 2s.

Les résultats sont présentés dans la figure 5.4. Ces résultats montrent que l'activité CPU observée est linéaire avec la demande de charge jusqu'à un certain point. Cette valeur d'observation atteint bien 200% pour l'utilisation de deux cœurs CPU à une charge demandée de 9500 itérations par seconde. La valeur maximale de charge CPU pour quatre cœurs (c'est-à-dire, 400% de temps CPU) n'est pas atteinte pour l'évaluation sur 4 cœurs, en effet pour une demande de 19000 itérations par seconde la charge n'est que de 392% du temps CPU et décroit si on demande une charge plus importante.

Cette figure présente en plus de l'activité CPU observée les taux d'erreurs observés par les stresseurs lors de l'exécution de la demande de charge. Ces taux sont très faibles (moins de 1 ms de retard par seconde d'exécution) jusqu'à ce que la demande de charge atteigne les capacités maximales des stresseurs.

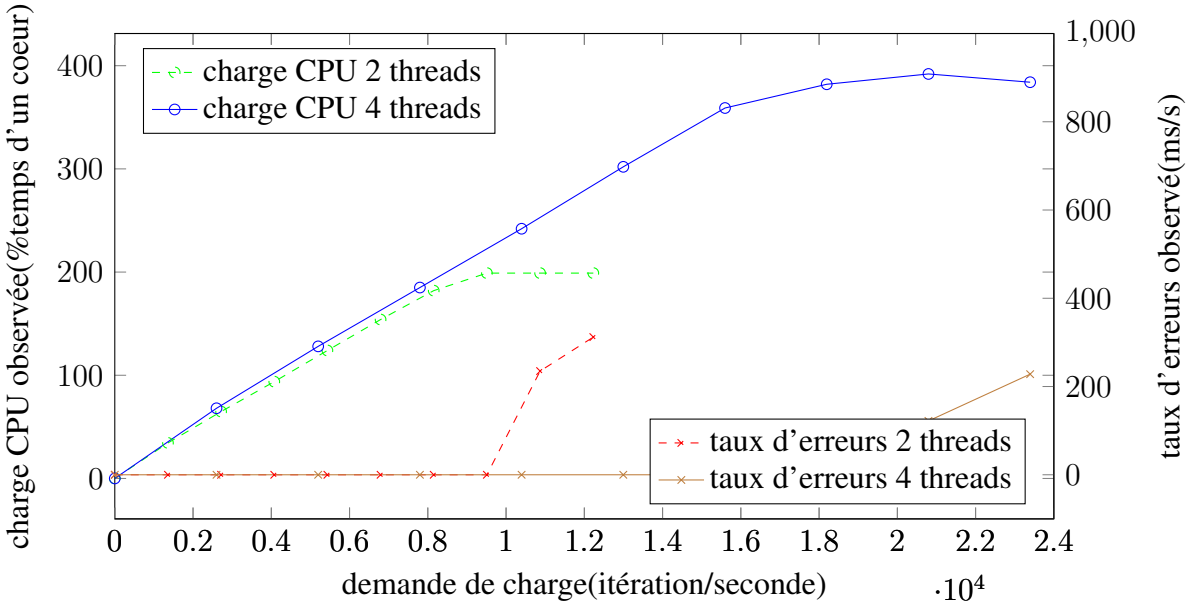


FIGURE 5.4 – Utilisation en pourcentage du temps CPU d'un cœur induite par le stresseur CPU.

La charge observée n'atteint pas les 400% de temps CPU d'un cœur qui seraient attendus par l'utilisation d'un processeur quadri-core, nous supposons que les mécanismes de synchronisation utilisés pour la répartition des tâches dans le stresseur réduisent l'efficacité de celui-ci. Aussi, l'utilisation sur plusieurs cœurs nous semble n'être possible que pour une utilisation faible du stresseur CPU.

Malgré cette singularité, cette évaluation valide la capacité de notre implémentation à injecter une charge constante de CPU en relation linéaire avec la demande.

stresseurs de disque et réseau

L'évaluation des stresseurs de disque et de réseau est similaire à celle du CPU, manipulant un stresseur de la ressource considérée. Après une première calibration du stresseur pour en déduire sa charge maximale, nous faisons varier sa demande de charge de 0 à 150% de cette charge maximale. Ce faisant nous observons la charge système correspondante sur le serveur manipulé ainsi que la surcharge CPU induite, après une période de stabilisation de 10s et une observation de l'activité pendant 5s.

Les résultats de cette évaluation sont présentés respectivement pour les stresseurs disque et réseau dans les figures 5.5 et 5.6.

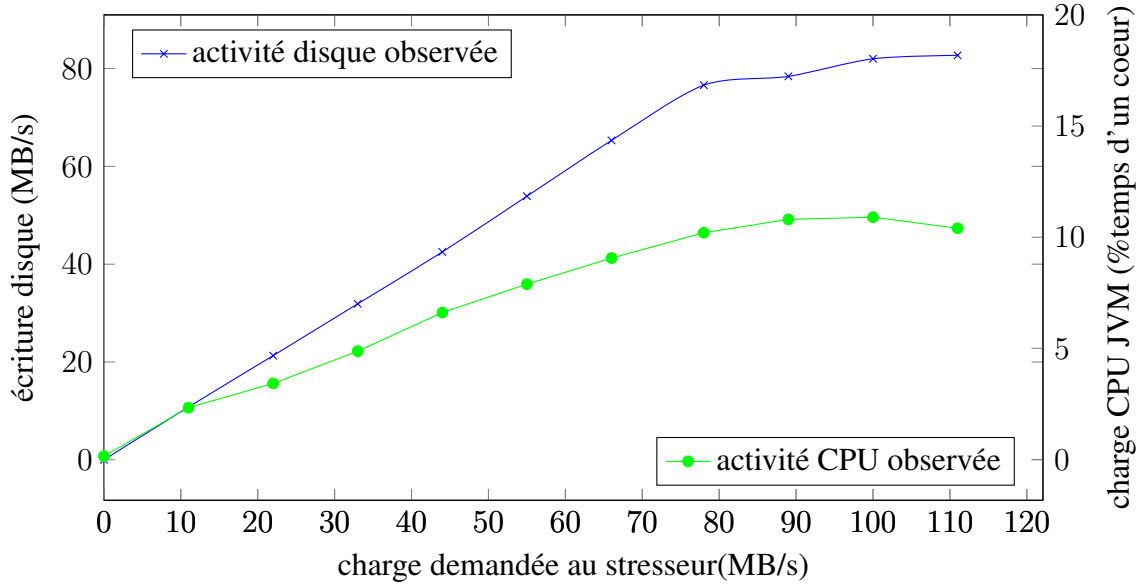


FIGURE 5.5 – Charges disque et CPU induites par le stresseur disque.

Le stresseur de disque est configuré à une granularité de 300ms, sa charge maximale déduite est de 82MB/s. L'activité d'écriture sur le disque est obtenu par l'utilisation de la librairie Sigar, donnant accès à la quantité de données écrites sur tous les disques du système.

Cette observation indique que l'observation de l'activité d'écriture sur le disque est bien linéaire avec la charge demandée, avec un taux d'erreurs entre la demande et l'observation de moins de 2% pour les demandes inférieures à la capacité maximale du

système. L'activité CPU croît cependant avec la demande de charge jusqu'à atteindre la valeur de 10% du temps CPU d'un cœur utilisé pour l'exécution de l'injection de charge. Cette valeur importante de surcharge CPU doit être prise en compte lors de l'injection de charge disque dans un centre virtualisé.

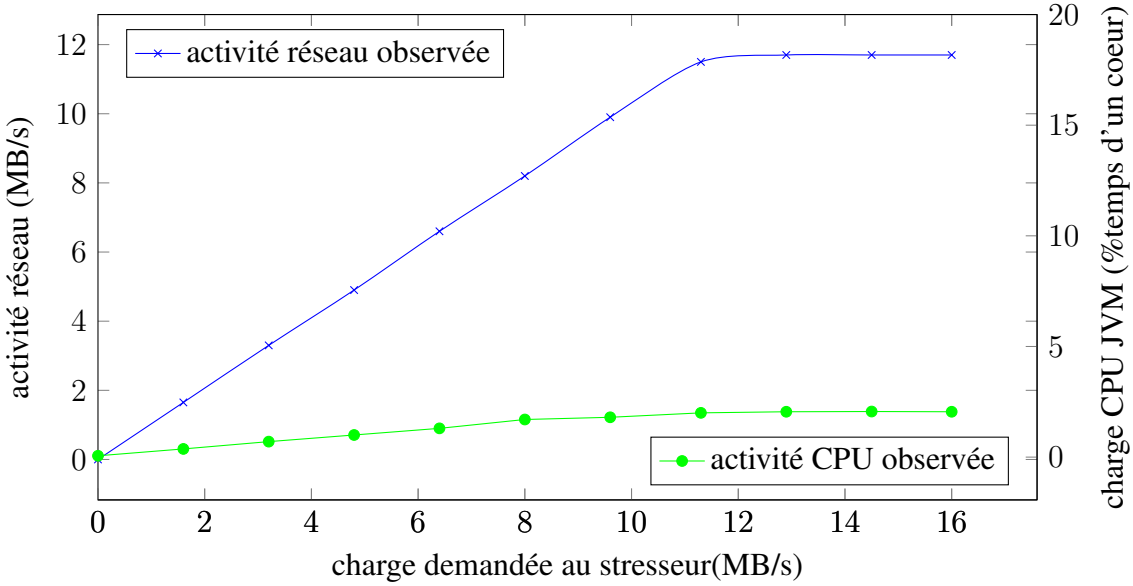


FIGURE 5.6 – Charges réseau et CPU induites par le stresseur réseau.

Le stresseur réseau est configuré à une granularité de 200ms, sa charge maximale déduite est de 10.7MB/s. L'observation de l'activité réseau se fait grâce à la librairie Sigar et permet de déduire pour une interface réseau le nombre d'octets émis sur cette interface, de la même manière que la commande `ifconfig eth0` par exemple sous linux. Cette observation du réseau coûte quelques opérations CPU, qui ne sont pas visibles dans notre expérimentation.

D'après notre évaluation, la charge réseau observée sur le système est bien linéaire avec la demande, à un taux d'erreurs de précisément $+2.5\%$ jusqu'à la charge maximale du stresseur. Ce taux d'erreurs peut être due à la présence des entêtes UDP. De fait, nous utilisons des datagrammes de taille fixe, induisant une surcharge réseau induite par les en-tête proportionnelle au nombre de datagrammes envoyés. De la même manière, les résultats indiquent une surcharge CPU linéaire avec la charge réseau demandée, jusqu'au seuil de capacité du stresseur. Cette surcharge atteint 2% du temps CPU utilisé par un cœur, ce qui est faible au regard de l'activité CPU usuelle d'un serveur avec une activité réseau importante. Cependant cette charge étant linéaire avec la demande de charge réseau, il peut être nécessaire de configurer plus finement le stresseur réseau

pour utilisation sur des serveurs à capacité réseau plus importante.

Cette évaluation confirme que nos implémentations de stresseurs correspondent à nos besoins malgré une surcharge CPU non négligeable.

Observation des erreurs

Dans cette évaluation nous évaluons la capacité du stresseur CPU à déterminer les surcharges. Pour cela nous utilisons un stresseur de CPU dans une VM enregistrée sur un registar, puis nous exécutons un scénario de montée progressive en charge sur ce registar. Durant le scénario nous mémorisons grâce aux outils de STRESSCLOUD les taux d'erreurs du stresseur toutes les 100s, puis à partir de ces valeurs d'erreur et de la demande de charge nous en déduisons la capacité CPU maximale du serveur. Le serveur utilisé est un HP Z200, le système d'exploitation Ubuntu, le stresseur n'utilise que 1 cœur.

Nous utilisons le scénario suivant :

```
release();base=2000;period=2*60
vm1=require(null)
vm1.ping(100)
lb=loadBase(period, base)
lb.to(vm1.cpu, 1,2,3,2.5,2.5,3.5,5)
after(period*7+1);loads().toCSV()
```

Ce scénario demande la réservation d'une VM dans le centre et lui spécifie une observation des activités à faire toutes les 100s au maximum. Puis il crée un modèle de charge, basé sur une valeur de charge de 2000 et une période d'exécution de 2 minutes, qu'il injecte dans le stresseur CPU de la VM en spécifiant une liste de 7 multiplicateurs de charge. La charge CPU du stresseur est donc successivement spécifiée à $2000 \times multiplicateur$ durant 2 minutes à chaque fois. Après une période de $7 \times 2min + 1min$, le scénario affiche les résultats d'exécution des charges.

Ce résultat est présenté sous forme de graphique dans la figure 5.7. Ce graphique montre, en fonction du temps, d'une part la charge CPU demandée au stresseur, d'autre part le taux d'erreurs (en %) observé par le stresseur durant cette demande de charge, et finalement, en utilisant ces deux valeurs, la capacité maximale de charge, déduite par la formule $capa = charge \times \frac{100 - erreurs_{prct}}{100}$.

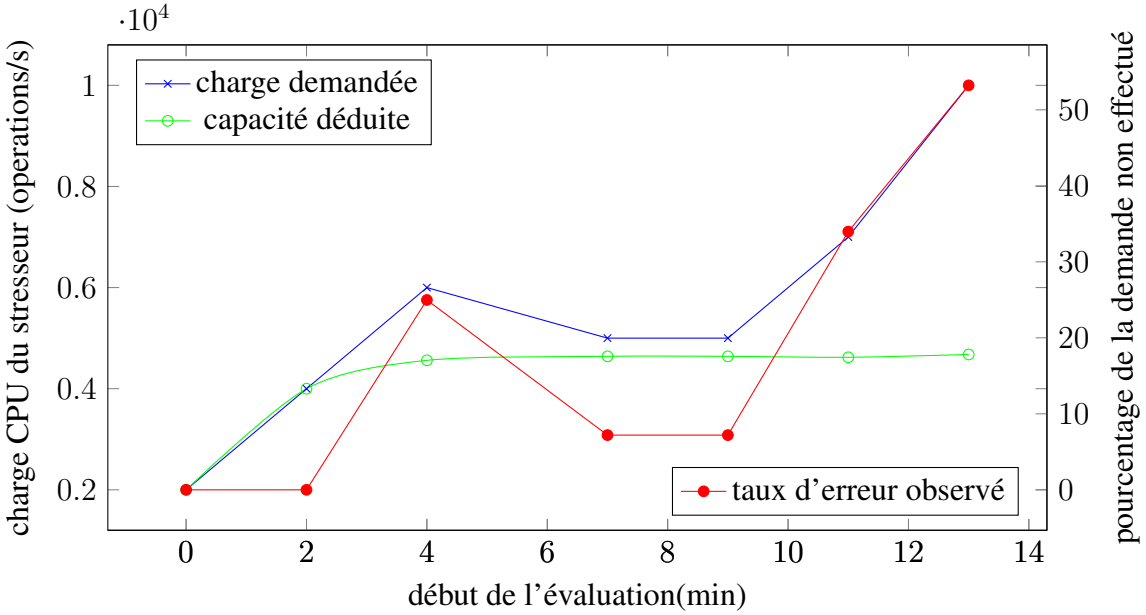


FIGURE 5.7 – demande de charge CPU, taux d'erreurs et capacité CPU déduite.

Ce graphe montre que le taux d'erreurs observé par le stresseur est croissant avec la charge demandé à celui-ci. Il est nul quand la charge charge est inférieure à un seuil donné. De plus, nous observons que la charge maximale déduite est bien constante tant que la charge demandée est supérieure au seuil de charge maximal. D'après cette évaluation, l'observation des taux d'erreurs dans un stresseur est un indicateur fiable de l'exécution effective du scénario.

Une fois que nous nous sommes assurés que nos stresseurs d'une part injectent bien une activité en relation avec la demande de charge dans la ressource système du serveur, et d'autre part observent correctement les taux d'erreurs de leur charge, nous nous intéressons à la capacité de modéliser la consommation électrique d'un serveur en fonction de l'activité de ses ressources.

5.4.2 Modélisation du coût électrique

Nous utilisons STRESSCLOUD pour observer les variations de la puissance électrique consommée par un serveur en fonction de l'activités des ressources de celui-ci. Pour cela nous exécutons un scénario où, pour un groupe de VM hébergées sur un même serveur hôte, nous injectons des charges croissantes dans plusieurs ressources successives. Durant l'exécution de ce scénario, nous mesurons la consommation électrique du serveur afin de la mettre en relation avec les activités des ressources.

Le serveur utilisé est un Dell Poweredge M100e, les ressources prises en compte sont le CPU, le réseau et les disques. Le résultat est présenté dans la figure 5.8

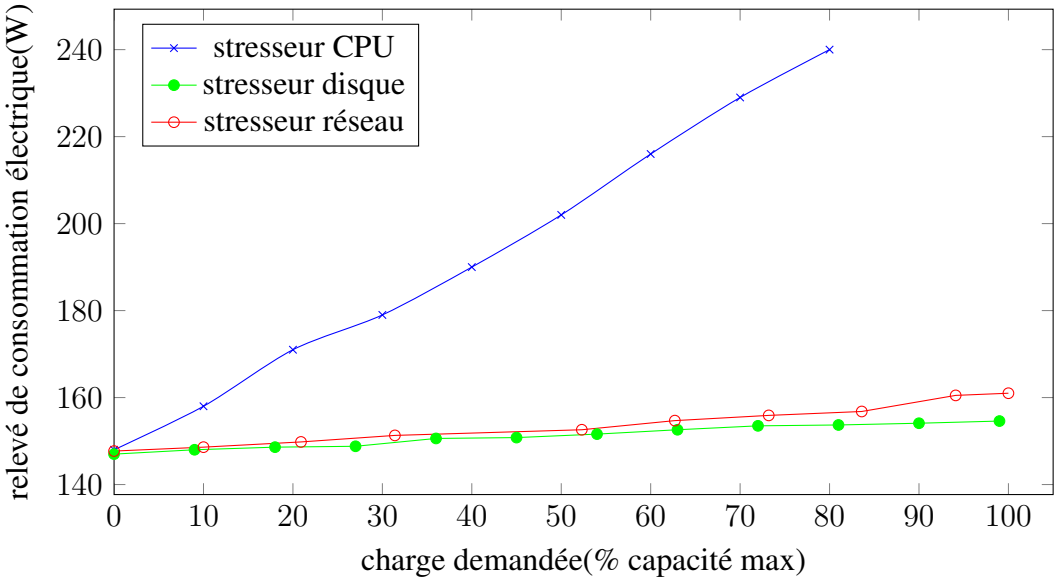


FIGURE 5.8 – Consommation électrique du serveur selon les charges demandées.

Pour des soucis de clarté nous avons d'abord calibré les stresseurs et utilisons dans cette figure les pourcentage d'utilisation des stresseurs par rapport à leurs capacités ainsi calibrées. Cette figure montre que la consommation électrique du serveur est croissante avec chaque ressource, cependant la ressource ayant le plus grand impact est la ressource CPU avec une augmentation de 92W, suivi par le réseau et le disque avec des augmentations de respectivement 13W et 7W.

Cette évaluation n'a pas pour but de déduire un modèle de consommation, qui nécessiterait le croisement des observations des charges système et l'utilisation de plusieurs VM pour charger tous les cœurs CPU. En effet, comme décrit les stresseurs disque et réseau induisent une activité CPU non négligeable, activité qui est responsable d'après cette évaluation d'une part importante (au moins 82%) de la consommation maximale. Il faudrait donc observer la surcharge CPU induite par ces stresseurs pour avoir une modélisation plus correcte de la consommation du serveur.

Cette évaluation montre que notre outil nous permet de produire des traces utiles de la consommation électrique pour un serveur. Il est en particulier utile pour obtenir ces traces à des niveaux d'activité qui peuvent ne pas être atteints dans une utilisation standard des serveurs. Ainsi, STRESSCLOUD permet de déduire des paramètres de la consommation électrique d'un serveur à partir de l'utilisation des ressources de ce dernier et, par sa capacité à manipuler plusieurs VM simultanément, d'obtenir des traces d'activité électrique au niveau d'un centre.

5.4.3 Évaluation du langage

Après nous être assuré de la capacité de STRESSCLOUD à injecter une activité ressource dans une VM, nous évaluons sa capacité à décrire et exécuter des scénarios manipulant plusieurs VM.

L'évaluation de la capacité descriptive se focalise sur la capacité d'un utilisateur de STRESSCLOUD à exprimer une série de charge, avec les synchronisations temporelles et inter-VM, tandis que l'évaluation d'exécution s'assure que des scénarios d'activité usuels sont bien perçu par les systèmes de gestion d'un centre. Nous déterminons donc des scénarios à produire, les traduisons dans STRESSCLOUD et les exécutons.

Scénario de type webserver

Notre premier scénario a pour objectif de produire une activité dans le centre proche de celle qu'aurait un centre hébergeant un service de type serveur internet 3-Tiers.

Dans cette classe d'infrastructure virtualisée, une VM du premier tiers (T1) reçoit des requêtes clients, les délègue à une des VM du deuxième tiers (T2) qui sont responsables du traitement des requêtes, tandis que les VM du troisième tiers(T3) se chargent du stockage des données et de leur obtention pour le traitement des T2 tiers. Nous choisissons de mettre en relation le nombre de requêtes arrivant à T1 avec l'activité CPU de T1, l'activité réseau induite des T2 vers les T3 et leur activité CPU, l'activité CPU et disque des T3. Le stockage des T2 est reparti équitablement entre les VM de T3 : pour simplifier le processus, chaque T2 est associée à une seule T3.

Nous faisons croître la charge totale du centre pendant une heure puis la faisons décroître pendant une demi heure. Si les charges des centres virtualisés de type webserver ont plutôt une période d'une journée, nous considérons qu'il est inutile d'exécuter un scénario sur une aussi longue période. Les valeurs d'accroissement de charge sont spécifiées dans une variable `loads` qui correspond aux multiplicateurs à appliquer à la charge maximale. Nous faisons un premier calibrage pour déterminer les capacité CPU des VM.

Le scénario complet est le suivant :

```
release()
t2nb=20;t3nb=2;
period=60*6;
netT2=10000;diskT3=netT2*t2nb/t3nb
loads=[0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1,0.9,0.8,0.7,0.6,0.5]

t1=require(null)
t2=require(null, t2nb)
t3=require(null, t3nb)
t2.eachWithIndex{it, i->it.net.target=t3[i%t3nb].ip}
```

```
lbT2cpu=loadBase(period, 300)
lbT2net=loadBase(period, netT2)
lbT3cpu=loadBase(period, 100)
lbT3disk=loadBase(period, diskT3)

loadBase(period, 100).to(t1.cpu,loads)
t2.each{lbT2cpu.to(it.cpu, loads);lbT2net.to(it.net, loads)}
t3.each{lbT3cpu.to(it.cpu, loads);lbT3disk.to(it.disk, loads)}
```

Ce scénario montre que STRESSCLOUD propose des mécanismes nécessaires à la description de scénario complexes :

- La gestion des variables, ici les charges maximales disque et réseau pour les T3 et T2.
- La mise en relation des variables des éléments manipulés. *eg.* `diskT3=netT2*t2nb/t3nb` et `it.net.target=t3[i%t3nb].ip`) mettent respectivement en relation la charge disque de base des T3 avec la charge réseau de base des T2 par un produit en croix, et l'affectation de la cible réseau des VM de T2 à une des VM de T3en les répartissant de manière homogène.
- Les opérations sur les collections, *eg.* `t2.each{<charge les VM de T2>}` qui permet d'ajouter une série de charge CPU et disque à toutes les VM de T2 en une seule ligne.

La capacité de séquencement temporelle des activités n'est pas suffisante : il faut aussi permettre le séquencement de travaux dans le cas où les ressources sont chargées au maximum de leur capacité.

Scénario de type NASGrid

La suite NASGrid [FVdW02] propose des scénario de banc d'essai pour grille de calcul, principalement axés sur les activité CPU et à forte charge de calcul (type HPC). Nous nous sommes intéressés à l'expression du scénario le plus simple de cette suite, le scénario ED[1].

Dans ce scénario ED, les VM sont réparties en trois tiers : la VM du premier tiers (T1) effectue une décision de répartition des tâches parmi les VM du deuxième tiers (T2) et démarre ainsi leurs activités. La VM du troisième tiers (T3) attend que les T2 aient effectué leurs tâches pour agréger leur résultat.

[1] voir ED : http://www.nas.nasa.gov/publications/npb.html

Nous exprimons ce scénario de la façon suivante :

```
amount=500000; tier2size=2
tier1=require(null);tier2=require(null, tier2size);tier3=require(null)

tier1.cpu+amount
tier2.each{it.after(tier1).cpu+3*amount}
tier3.after(tier2).cpu+amount

after(tier3);works().toCSV()
```

Ce scénario très simple affirme la capacité du système à décrire correctement la planification d'activités de type HPC. L'utilisation d'un langage de script autorise de plus les utilisateurs d'une installation de STRESSCLOUD à créer des modèles de scénarios, modifiant à la demande des paramètres de charge ou de nombre de VM impliquées. Ainsi dans ce scénario le nombre de VM présentes dans T2 est une variable et peut facilement être modifiée pour passer de 2 ici à 100 VM tout en gardant les mêmes valeurs de charge.

L'exécution de ce script sur un centre controlé par un SGIV produit deux résultats : les observations d'activité des stresseurs et les observations d'activité du SGIV. Cette exécution a été effectuée sur un centre géré par un VMWare VCenter 5.0, les VM exécutant des OS de type Ubuntu server, hébergées sur un seul serveur de type Dell Poweredge M100e .

Les observations des stresseurs sont présentées dans le tableau 5.1, indiquant pour chaque demande de travail la ressource manipulée, le temps de début et celui de fin de l'exécution de ce travail, et finalement la quantité de travail demandée. Ce tableau est créé par STRESSCLOUD lorsque nous lui demandons à la fin d'un scénario la liste des travaux : `works().toCSV()`.

resource	start	end	work
vm4.cpu	761	944	500000.0
vm3.cpu	183	761	1500000.0
vm2.cpu	183	761	1500000.0
vm1.cpu	0	183	500000.0

TABLE 5.1 – Observation des travaux ajoutés et de leurs dates de début et de fin

Les observations du SGIV contiennent, en fonction du temps, l'activité CPU des VM présentes sur le serveur. Ces observations sont présentées dans la figure 5.9.

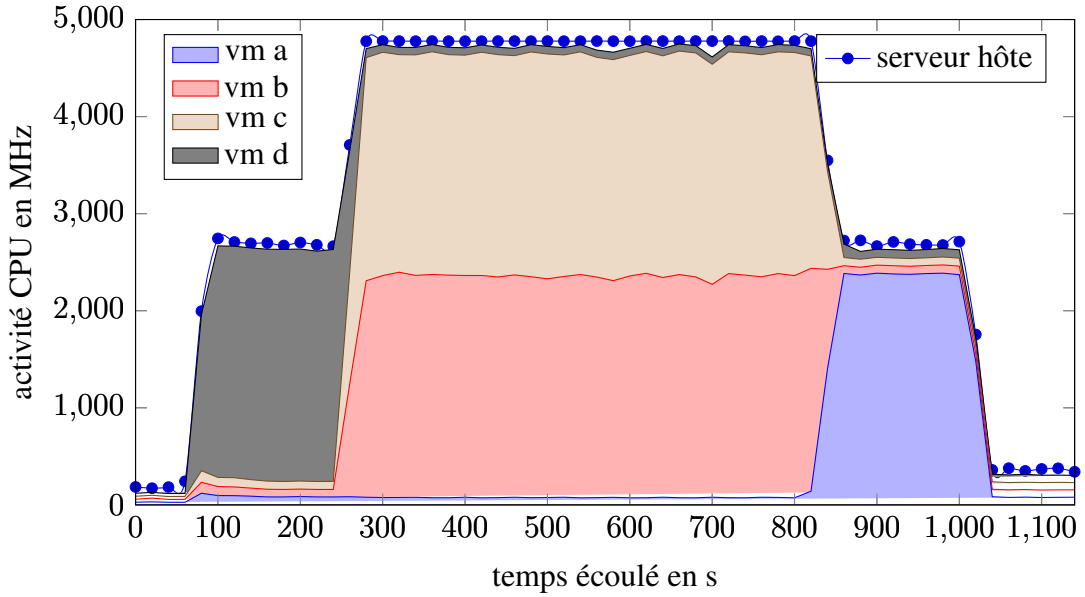


FIGURE 5.9 – Observation de l'activité CPU induite par l'exécution du scénario NAS-Grid.

Cette figure étant issue de l'observation du SGIV, les noms des VM ne correspondent pas au nommage qui est fait dans le scénario. En effet travaillant dans un IaaS nous n'avons pas accès depuis le registar au système de gestion de ce dernier.

De cette figure on observe que la charge des VM suit bien un premier pallier où une seule VM a une activité CPU importante, suivi d'une étape où deux autres VM accèdent régulièrement au CPU à leur disposition, et enfin une troisième étape où seule une quatrième VM a une forte activité CPU. En mettant en relation les valeurs temporelles indiquées dans le tableau 5.1 et celles de la figure 5.9, nous pouvons affirmer que les temps sont les mêmes à un décalage de 80s près correspondant au temps d'attente entre l'observation du centre et l'exécution du scénario. De plus, cette évaluation nous permet de constater que l'activité CPU du serveur hôte n'est pas exactement égale à la somme des activité de ses VM, mais toujours légèrement supérieure par quelques MHz, et ce quelque soit la charge des VM hébergées.

Cette évaluation sur un centre réel montre que ce scénario est correctement exécuté sur le centre, puisqu'il est bien perçu par le SGIV. Elle permet aussi d'affirmer que les performances enregistrées par les outils d'observation inclus dans STRESSCLOUD sont conformes à la réalité.

5.5 Conclusion

Dans ce chapitre, nous avons présenté un outil d'injection de scénario d'activité complexe dans un centre virtualisé. Cet outil, nommé STRESSCLOUD, permet une injection contrôlée d'activités, limitées dans le temps ou en quantité d'actions, parmi les ressources système accessibles aux VM manipulées. STRESSCLOUD synchronise les activités requises par des expressions temporelles ou de succession et permet d'observer l'exécution effective de ces activités.

Nous avons montré que le langage de scénario utilisé pour STRESSCLOUD permet de décrire des scénarios complexes. Mais aussi que nos implémentations de stresseurs assurent une injection d'activité constante qui peut être utilisée pour la modélisation énergétique d'un serveur. Nous avons finalement montré que l'exécution effective d'un scénario est bien observée par les mécanismes de retour mis en place, permettant d'évaluer les capacités des serveurs et la réactivité des SGIV sans pour autant avoir accès à ces derniers.

L'outil est générique et il serait assez simple de développer de nouveaux stresseurs, comme par exemple pour une carte graphique. Le système central chargé de contrôler les activités, le registar, pourrait être un point bloquant pour le passage à l'échelle du système. Cependant, sur les évaluations menées, nous n'avons détecté aucune saturation, mais nous pensons tout de même réaliser une expérience large échelle sur Grid5000

Nous pensons également développer un module complémentaire à STRESSCLOUD ayant pour objectif de rejouer des traces d'activités collectées dans des centres de données réels. Pour cela, un travail d'analyse et de synthèse des traces doit être effectué, suivi par le pilotage de l'outil via l'interface de type REST déjà développée.

6

Expérimentation

Sommaire

Dans ce chapitre, nous évaluons les capacités descriptives ainsi que les performances d'OPTIPLACE. Cet outil étant une modification d'ENTROPY, nous comparons tout d'abord dans la section 6.1 les problèmes adressés avec ceux de l'original. Nous évaluons ensuite la capacité d'OPTIPLACE à incorporer des travaux et modèles externes dans la

section 6.2, puis en section 6.3 nous comparons les performances de la vue énergétique d'OPTIPLACE à d'autres travaux.

6.1 OPTIPLACE et ENTROPY

Cette section s'attache à décrire le cadre d'utilisation de OPTIPLACE OPTIPLACE est un outil d'administration de centre virtualisé dont le cœur est issu des travaux ENTROPY. Si ENTROPY et OPTIPLACE s'attachent tous deux à résoudre un problème de placement des VM, les problèmes adressés varient d'une solution à l'autre. Nous commençons donc par comparer les cadres d'utilisation, et donc les fonctionnalités, que ces deux solutions considèrent.

Nous ne nous intéressons pas ici directement au système de vues, qui est un moyen de modéliser des problèmes supplémentaire par dessus le cœur d'OPTIPLACE. Ce système permet d'enrichir ce cœur mais n'ajoute pas d'informations spécifiques en tant que tel. Nous ne nous attachons ici qu'aux problèmes adressés par le cœur d'OPTIPLACE, muni de sa vue HA, et ENTROPY.

6.1.1 Gestion des actions administrateur

Comme indiqué dans le chapitre 4, ENTROPY propose, pour un centre donné, non seulement une amélioration de la configuration virtuelle de ce centre, mais aussi un agenda d'actions à exécuter. Cet agenda spécifie des actions virtuelles constituant une transition de la configuration initiale vers la configuration cible.

La construction de cet agenda, intégrée dans le problème de placement, assure la capacité de l'administrateur à atteindre la configuration cible. Elle permet aussi de définir une borne supérieure du temps de reconfiguration nécessaire. Pour cela, ENTROPY nécessite des modèles de durée des actions possibles. Lors de la résolution du problème de placement, ENTROPY intègre les actions possibles et leur durée grâce à ces modèles de durée.

Désactivation du système de planification

Si cette planification est intéressante d'un point de vue algorithmique et recherche de solution, nous avons estimé que son utilisation était trop contraignante. En effet, la fiabilité de cette planification est conditionnée à un travail de modélisation des durées des actions dans le centre. Cette modélisation nécessite des études approfondies de l'architecture virtuelle du centre et peut ne pas être déductible à une précision suffisante.

De plus, la modélisation de chaque action possible dans le centre implique une utilisation accrue de la mémoire et du CPU lors de la résolution d'un problème. En effet, chaque migration possible d'une VM vers un serveur doit être représentée et évaluée

dans le solveur, et le délai avant initialisation d'une action doit prendre en compte toutes les actions possibles.

Finalement, cette planification impose des contraintes sur la configuration cible solution du problème. Ces contraintes peuvent produire des effets de bord lors de la résolution d'un problème en réduisant les solutions possibles. Les développeurs voulant modifier certains comportements doivent alors prendre en compte ces contraintes, quand bien même celles-ci ne concernent pas leurs travaux.

Pour ces raisons, OPTIPLACE n'intègre pas cette fonctionnalité de planification dans son cœur. Cette perte de fonctionnalité du cœur peut cependant être compensée par le développement d'une vue dédiée dans OPTIPLACE. En effet, le système de vue permet par exemple de créer des variables de début et de durée d'action associées à chaque VM, puis de contraindre ces variables selon le serveur hôte d'une VM, et finalement déterminer une durée totale de l'agenda. Ainsi OPTIPLACE ne supporte pas cette fonctionnalité mais permet de l'intégrer.

De plus, notre architecture modulaire permet de changer l'implémentation d'une même fonctionnalité, par l'isolation de celle-ci dans une vue. Il est alors possible de développer non pas une, mais plusieurs implémentations de cette planification, afin de comparer la recherche de solution en utilisant chacune. Ainsi, OPTIPLACE permet aux développeurs de créer un système de planification, optionnel, dans un problème de placement, mais aussi de modifier et comparer différentes implémentations d'un tel système.

Réduction des actions administrateur

ENTROPY proposait non seulement des actions de migrations des VM, mais aussi des actions de démarrage et d'extinction des VM et des serveurs. ENTROPY déterminait donc, pour chaque action possible dans le centre, lesquelles étaient utilisées. Nous avons supprimé ces actions de démarrage et d'extinction pour deux raisons.

Premièrement, pour une raison de responsabilité accordée à OPTIPLACE. En effet, si la migration de VM est une action éprouvée dans les centres virtualisés, l'extinction arbitraire de VM ou serveurs peut poser des problèmes inattendus. Par exemple, certains serveurs peuvent ne pas permettre, pour des raisons de configuration ou de matériel, un démarrage à distance. Aussi, un administrateur préférera démarrer automatiquement les VM, laisser celles-ci décider de leur propre extinction, et choisir quels serveurs allumer ou éteindre.

Deuxièmement, pour une raison de performances. La modélisation de chaque action possible demande une quantité de mémoire importante et des contraintes supplémentaires. Ainsi, l'extinction d'un serveur nécessite la migration de chaque VM de ce serveur, tandis que le démarrage d'un serveur nécessite des conditions particulières, spécifiées par l'administrateur. Si ces contraintes ne demandent pas des temps de calcul importants lors de la résolution, elles peuvent cependant induire des heuristiques en

erreur.

Ce retrait des actions de démarrage et d'extinction n'est pas pénalisant pour la gestion du centre. En effet, le démarrage de serveur n'était nécessaire que lorsque la charge totale du centre était trop importante, ou qu'il n'était plus possible de démarrer une nouvelle VM. Ce besoin peut être pris en compte par des règles de réservation de ressources supplémentaires.

L'extinction de serveurs, quant à elle, n'est plus une action mais une possibilité déduite par OPTIPLACE. Celle ci est utilisée dans le modèle énergétique, pour annuler la consommation d'un serveur qui peut être éteint. Par défaut, les serveurs d'un centre ne peuvent être éteints, seuls des SGIV gérant cette possibilité pourront introduire cette option.

6.1.2 Évolution du modèle de ressources

La satisfaction des besoins en ressources des VM, sous des règles variées spécifiées par l'administrateur, était au cœur du problème adressé par ENTROPY. Cette notion de satisfaction a été étendue dans OPTIPLACE.

D'une part, nous permettons à des développeurs d'intégrer, dans des vues, de nouvelles ressources à prendre en compte dans l'allocation des VM sur les serveurs. D'autre part, cette notion de ressource est visible dans l'architecture, ainsi une vue peut remplacer la spécification d'une ressource par une autre. OPTIPLACE permet donc la prise en compte de multiples ressources spécifiées dynamiquement par les vues.

Si cette spécification dynamique des ressources permet d'étendre facilement le problème, elle réduit cependant la facilité de développement des heuristiques. En effet, une heuristique ne peut plus s'appuyer sur une connaissance statique des ressources utilisées dans le centre, et doit dorénavant sélectionner les ressources parmi celles considérées dans le problème. Bien entendu, si les heuristiques sont développées dans un cadre bien précis, ces ressources sont connues et cette variabilité des ressources n'est plus un problème.

Enfin, l'intégration de ces ressources dans un problème, traduisant des données d'utilisation et de capacité en variables contraintes dans un solveur, est aussi paramétrables dans OPTIPLACE. Ce paramétrage permet non seulement de sélectionner des implémentations adaptées aux ressources utilisées, mais aussi d'adapter la résolution de problèmes à la taille de ceux-ci, participant ainsi au passage à l'échelle de l'outil.

6.1.3 Passage à l'échelle pour des problèmes de grande taille

Si ENTROPY cherchait, dans une certaine limite de temps, une solution optimale à un problème de placement, cette recherche optimale devient impossible pour des problèmes de grande taille. En effet, l'espace des solutions potentielles à explorer devient trop im-

portant, requérant alors la recherche de solutions non plus optimales mais suffisamment bonnes.

Ainsi OPTIPLACE, de même qu'ENTROPY, permet de demander une solution satisfaisante, et non plus optimale, dans un problème. Si la fonction de timeout lors d'une recherche n'est plus présente, nous avons implémenté une fonctionnalité de réduction de l'objectif, qui évite la recherche de solutions supplémentaires quand une solution suffisamment bonne est trouvée. Cette fonctionnalité réduit la taille de l'arbre de recherche, mais pour des problèmes de grande taille elle n'est pas suffisante, et il faut alors s'arrêter à la première solution déduite par la stratégie de recherche.

Les stratégies de recherches, ou heuristiques, sont visibles dans OPTIPLACE. Un développeur peut alors créer de nouvelles heuristiques pour résoudre un problème, et sélectionner, parmi les heuristiques présentes, celles adaptées non seulement au domaine de la vue, mais aussi à la taille du problème. Ainsi, des heuristiques moins coûteux en calculs peuvent être sélectionnés pour les problèmes de grande taille, tandis que des heuristiques très complexes seront utilisés pour des problèmes de petite taille. Cette fonctionnalité, qui consister ainsi à adapter l'implémentation du problème, est proche de celle permettant de spécifier l'intégration des contraintes de ressource.

Nous avons ainsi défini l'évolution fonctionnelle d'OPTIPLACE par rapport à ENTROPY. Le système de vues permet aux développeur d'intégrer par exemple les résultats de travaux portants sur la gestion des centres virtualisés.

6.2 L'intégration de nouvelles contraintes et paramètres

Une fois que nous avons comparé les cadres d'OPTIPLACE et ENTROPY, nous évaluons la capacité d'OPTIPLACE à intégrer des modèles externes. Pour cela, nous proposons des vues permettant d'intégrer des travaux de gestion des centres virtualisés.

6.2.1 Modèles de SLA de l'Impact Lab

Les travaux de l'impact lab [AMVG11] se placent dans un centre de type ou PaaS, où un client demande l'exécution d'applications spécifiques. Cette demande s'accompagne d'un contrat de service, ou SLA, stipulant des coûts supplémentaires pour l'administrateur du centre si les conditions d'exécution ne sont pas satisfaites.

Dans ces travaux, cette équipe a travaillé sur les SLA de délai de réponse pour un serveur web. Cette équipe a déterminé que ces SLA peuvent être exprimé en fonction de la charge CPU du serveur qui exécute la VM de l'application. Ces SLA peuvent donc être pris en compte au niveau administrateur en majorant l'activité CPU du serveur hôte de la VM.

Nous proposons trois règles supplémentaires dans la vue HA pour permettre l'intégration de tels SLA dans OPTIPLACE. Ces trois règles intègrent des analyses différentes

de ce problème de SLA.

La première règle, nommée *MarginHostCPU* intègre directement cette notion dans le problème à résoudre. Elle considère une VM et un taux d'utilisation maximal sous forme de pourcentage. Lorsqu'elle est injectée, elle contraint la charge CPU du serveur hôte de cette VM à être inférieur au taux indiqué. Cette règle injecte de manière directe la notion de respect de SLA, cependant elle est très coûteuse en terme de calcul CPU. En effet, cette règle nécessite pour chaque serveur une variable de charge et ajoute une contrainte conditionnelle sur ces variables selon le serveur hôte de la VM. Si les variable de charge des serveurs sont mutualisées parmi plusieurs règles, ce n'est pas le cas des contraintes conditionnelles. Or, ces contraintes étant liées aux charges de chaque serveur, elles devront effectuer un traitement à chaque fois qu'une VM sera placée sur un serveur du centre durant l'exploration de l'arbre de recherche. Cette règle est alors trop coûteuse pour être utilisée sur un problème de grande taille.

Lorsque le nombre de VM sur lesquelles portent de tels SLA est suffisamment important, l'administrateur peut spécifier le regroupement de ces VM sur des serveurs dédiés. Dans ce cas, les VM sont astreintes à un groupe de serveurs, pour lesquels les activités des ressources sont majorées. Ainsi, un nombre réduit de serveurs auront leur activité limitée et idéalement les VM ayant les mêmes seuils seront sur le même serveur. Cette interprétation est intégrée par la règle nommée *LazyNode*, qui ne crée aucune contrainte ni variable dans le problème modélisé. Elle considère un serveur, une ressource et un taux d'utilisation maximal de cette ressource. Quand elle est injectée, cette règle majore simplement l'utilisation de ce serveur par ce taux. Du fait de ses empreintes mémoire et CPU négligeables, cette règle est adaptée à la résolution de problème de très grande taille, mais demande cependant une pré-analyse importante du problème.

Dans certains cas, une telle contrainte ne peut pas être assurée de manière systématique. Par exemple, lorsque les capacité CPU des serveurs sont trop faibles, c'est-à-dire que le centre a une charge trop importante. Plutôt que de spécifier des règles non viables ou ne rien faire, l'administrateur peut encore réduire les problèmes de SLA La dernière règle, nommée *ResourceOrder*, contraint les activités des ressources des serveurs considérés à suivre un certain ordre. Ainsi, lorsque l'administrateur a placé ses VM avec SLA sur un groupe de serveur, il peut demander à ce que ceux-ci aient par exemple une activité CPU inférieure au autres serveurs, réduisant alors les problèmes de rupture des contrats de service.

Ces trois règles, intégrées à la vue HA, permettent à l'administrateur d'intégrer les notions présentées dans les travaux de l'Impact, avec différents compromis de précision et temps de calcul. D'autres vues peuvent aussi intégrer des notions communes à différents travaux, comme le regroupement des VM et serveurs pour une facilité d'administration accrue

6.2.2 Vue de regroupement

Lorsqu'un administrateur veut manipuler un nombre très important de VM et serveurs, il devient fastidieux de spécifier des contraintes individuelles. Par exemple, lorsqu'un client fournit plusieurs VM, il est intéressant de placer celles-ci sur un groupe réduit de serveurs pour simplifier leur administration et leurs performances.

La vue de regroupement, non implémentée, permet à l'administrateur de créer des groupes de VM et de serveurs. Ce regroupement peut être fait de manière unitaire, en sélectionnant les noms des VM ou serveurs d'un groupe, ou bien par l'utilisation de filtres sur les noms de ces éléments.

Une fois les groupes créés par l'administrateur, sous la forme d'un paramétrage de la vue, ce dernier ajoute des règles de placement relatives au groupe. Si les première règles sont simplement des extensions de la vue HA à un modèle de groupe, nous proposons aussi des règles de priorité qui n'étaient pas présentes dans ENTROPY.

Les première règles concernent le placement des VM sur les serveurs. Les règles proposées dans ENTROPY sont étendues pour prendre en compte non des éléments du centre, mais des regroupement d'éléments. Ainsi, un groupe de VM peut être hébergé sur un groupe de serveur, ou sur le même groupe de serveur parmi plusieurs. De même, il est possible d'éteindre, pour des raisons de maintenance, un groupe de serveurs. Ces règles sont très simples et apportent simplement une couche de sucre syntaxique pour l'administrateur du centre.

Nous proposons de plus des règles d'allumage séquentiel de serveurs. Ces règles supposent qu'il est possible d'éteindre des serveurs inutilisés afin de ne les allumer qu'au besoin. Elles permettent à l'administrateur de spécifier une notion de priorité dans l'allumage des serveurs. Ces règles intègrent toutes le même principe : tous les serveurs d'un groupe prioritaire doivent être suffisamment utilisés avant que des serveurs d'un groupe non prioritaire ne puissent héberger de VM.

Ces règles considèrent donc deux groupes de serveurs, le premier prioritaire au deuxième, et une notion de seuil d'éveil, typiquement sous la forme d'un seuil d'utilisation. Tant que le seuil d'éveil n'est pas atteint sur le groupe prioritaire, alors les serveurs du groupe non prioritaires ne peuvent pas héberger de VM. Une telle règle peut pas exemple nécessiter au moins 5 VM sur chaque serveur du groupe prioritaire, ou une charge CPU supérieure à 50%.

Cette vue de regroupement n'a pas été implémentée par manque de besoin d'utilisation. Cependant, l'injection des premières contraintes est très facile par l'utilisation de la vue HA, tandis que les deuxièmes contraintes peuvent être facilement injectées en utilisant les fonctionnalités du solveur. En effet, cette proposition de vue apporte de la facilité d'utilisation à l'administrateur, tandis que d'autres vue intègrent des données externes, telles la notion de priorité CPU des applications.

6.2.3 Gestion des priorité CPU

Si notre modèle de ressource suppose les consommations de ressources des serveurs comme étant linéaires avec les consommations des VM qu'ils hébergent, d'autres modèles peuvent prendre en compte des relations plus complexes.

Citons comme exemple le *credit scheduler* de l'hyperviseur Xen qui permet de spécifier, pour chaque VM, une priorité d'accès au CPU ainsi qu'une limite de consommation du CPU. Ainsi, la consommation de CPU des VM placées sur un serveur dépend non seulement de leur demande, mais aussi de leur priorité et des consommations des autres VM hébergées sur ce serveur.

De même, le *tranparent memory sharing* de VMWare mutualise les pages mémoires de VM basées sur les mêmes OS et applications. Ainsi, si pour des raisons des performances plusieurs VM sont clonées sur le même hyperviseur, la réservation de la mémoire du serveur sera bien inférieure aux besoins de mémoire de ces VM.

Ces considérations sont très complexes et des modèles efficaces peuvent être très difficiles à déterminer et maintenir pour des développeurs. Aussi ces différents modèles ne sont pas présents dans nos travaux, bien qu'utilisés dans l'industrie. Cependant, des modèles de consommation de ressource étendus peuvent être créés dans des vues, afin de prendre en compte des modèles simplifiés. Ce travail de regroupement fait partie de nos travaux futurs.

De tels modèles d'allocation des ressources demandent des travaux importants de représentation et d'algorithmie, nous n'avons donc pas développé de vue les intégrant.

6.2.4 Activité et extinction des serveurs

Les travaux d'ENTROPY portaient sur la consolidation des serveurs d'un centre, en éteignant les serveurs inactifs suite à un placement judicieux de leurs VM. Cependant, l'administrateur d'un centre ne voudra pas éteindre tous les serveurs inactifs, afin de pouvoir démarrer au plus tôt une nouvelle VM ou de répartir plus rapidement la charge lors d'une hausse d'activité des services.

Nous proposons alors d'ajouter au modèle d'OPTIPLACE des notions d'autorisation d'extinction des serveurs. De telles notions considéreraient non seulement la capacité, physique ou logicielle, d'éteindre des serveurs du centre, mais aussi des autorisation d'extinction par l'administrateur du centre. En utilisant une vue appropriée, l'administrateur pourrait alors spécifier des règles d'extinction des serveurs, par exemple en demandant d'avoir au moins cinq serveurs actifs et non utilisés.

Une telle vue permettrait d'intégrer plus finement les modèles de consommation des serveur. En effet, dans l'état actuel de notre vue énergétique, l'administrateur doit spécifier de manière statique l'autorisation d'extinction d'un serveur. Un serveur ayant une autorisation d'extinction a ainsi une consommation nulle lorsqu'inactif, tandis qu'un serveur sans cette autorisation a une consommation minimale spécifiée par son modèle.

De tels considérations sont intégrables dans notre architecture, bien que nécessitant des travaux de recherche supplémentaires.

Nous avons étudié des développements possibles pour la gestion de centres virtualisés réels, mais bien qu'ayant pris en compte la capacité d'OPTIPLACE à intégrer des problèmes variés, le développement et l'intégration de ces problème n'était pas au cœur de notre thèse. Nous avons évalué dans cette thèse la capacité d'OPTIPLACE, muni de sa vue énergétique, à modéliser et améliorer un centre virtualisé, en considérant la consommation électrique de ses serveurs. Ces évaluations sont présentées dans la section suivante.

6.3 Résolution de problèmes énergétiques

La vue énergétique permet de modéliser les consommations de serveurs de manière générique. Cette généricité des modèles permet de nombreux cas d'utilisation, ainsi que des heuristiques qui peuvent être complexes et adaptés à des cas spécifiques. L'évaluation de la vue énergétique doit tenir compte de cette complexité.

Dans cette section nous comparons d'abord les performances de cette vue énergétique avec d'autres travaux portants sur la réduction de la consommation d'un centre, puis nous comparons les effets des différents modèles de consommation sur ces performances. Nous évaluons ensuite les effets de l'ajout de contraintes HA à des contraintes énergétiques, et finalement la capacité du système à considérer des ensembles de ressources plus importants.

6.3.1 Évaluation de la recherche de la première solution

Notre première évaluation compare la résolution d'un même problème en utilisant l'implémentation OPTIPLACE et en utilisant l'implémentation qui a été réalisée dans F4G. Nous comparons d'une part la consommation électrique de la solution déduite, d'autre part le temps nécessaire à la résolution du problème.

F4G[BBG+10] est un projet d'intégration de stratégies énergétiques dans des gestionnaires de centre virtualisé. Ce projet modélise la gestion énergétique d'un centre à différents niveaux, et intègre une telle modélisation sous forme de plug-in dans différents gestionnaires de centres. En particulier, un plug-in a été réalisé pour ENTROPY, considérant des consommation des serveurs du centre.

Pour cette évaluation nous devons considérer un centre qui puisse être représenté dans ces deux implémentation, car si F4G et OPTIPLACE permettent tous deux de prendre en compte des problématiques de consommation électrique dans un centre virtualisé, les approches, et donc les modèles de centre, sont différentes. Le modèle de centre que nous avons choisi pour cette première évaluation est modélisé de la manière suivante.

Le modèle S1 de serveur a une ressource CPU de capacité 10, une ressource RAM de capacité 10. La consommation électrique d'un tel serveur linéaire avec l'utilisation de sa ressource CPU, variant de 50 à 250W. Ainsi, chaque réservation de 1 ressource CPU par une VM augmente la consommation de ce serveur de 20W.

Le modèle S2 de serveur propose quant à lui 20 CPU et 20 RAM, et sa consommation électrique variant de 120 à 480W chaque réservation de 1 ressource CPU par une VM augmente cette consommation de 18W. Le modèle S1 est donc plus efficace que le S2 à charge CPU réduite, mais le modèle S2 est plus efficace que le modèle S1 si sa charge CPU atteint au moins 18.

Bien que la représentation d'un serveur dans F4G ne prenne pas en compte les ressources de celui-ci dans la modélisation de sa consommation, nous nous assurons que la même configuration virtuelle génère la même consommation électrique pour chacun des serveurs du centre en utilisant les deux implémentations. En effet le modèle électrique de F4G considère l'exécution d'une VM type sur chaque serveur pour en déduire les paramètres de consommation de celui-ci. C'est pourquoi nous considérons que toutes les VM présentes dans le centre sont du même modèle V, requérant 1 RAM et 1 CPU.

La configuration initiale d'un problème à résoudre comporte N serveurs de type S1 hébergeant chacun 3 VM de type V, ainsi que N serveurs de type S2 hébergeant chacun 6 de ces VM. Cette configuration comporte donc $2N$ serveurs et $9N$ VM.

Une fois la configuration initiale définie et les modèles de consommation électrique correctement paramétrés, nous soumettons les problèmes de placement correspondant à OPTIPLACE et F4G. Nous obtenons ainsi une configuration et une consommation cibles proposées par l'implémentation choisie. Cette consommation peut être obtenue de deux manières : en mode idle-off, les serveurs n'hébergeant aucune VM sont éteints pour ne consommer aucune puissance électrique, tandis que dans le mode idle-on, ces serveurs sont maintenus allumés et consomment donc une puissance de veille.

Les résultats de ces comparaisons, pour plusieurs tailles de centre virtualisé, sont présentées dans les figures 6.1 et 6.2. La première figure 6.1 présente les consommations électrique initiales et résultantes des algorithmes d'OPTIPLACE et F4G. Ces courbes indiquent la consommation en mode idle-on et idle-off des configurations cibles. Dans la configuration initiale les VM sont réparties sur tous les serveurs, ces deux modes de calcul déterminent donc la même consommation. Ces résultats montrent que notre implémentation permet, pour ce problème simple, d'obtenir les mêmes résultats que ceux obtenus par l'algorithme de F4G, que l'on permette ou non l'extinction des serveurs inactifs. Ce résultat est bien celui attendu car, dans ce problème, les contraintes de ressources et de placement ne vont pas à l'encontre du principe de packing, et permettent donc de placer toutes les VM sur un nombre réduit de serveurs du modèle S2.

La figure 6.2 présente en plus les temps de résolution requis pour ces évaluations. Elle montre que notre implémentation met moins de temps que celle de F4G pour résoudre le problème, allant jusqu'à nécessiter deux fois moins de temps. Nous déduisons

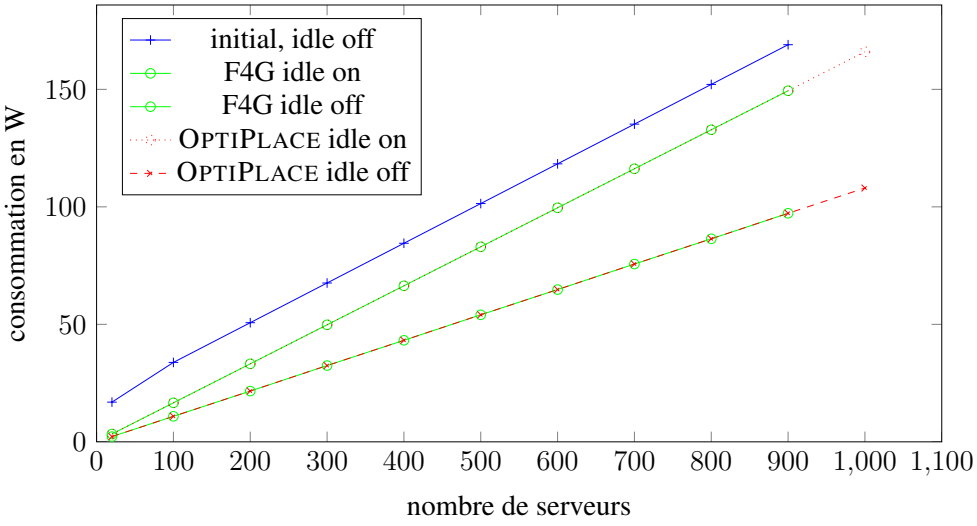


FIGURE 6.1 – Comparaison des consommations électrique de F4G et OPTIPLACE.

de ces résultats que notre implémentation, qui prend en compte des modèles génériques de consommation des serveurs et autorise l'utilisation de vues externes dans un problème, ne souffre pas de cet ajout de fonctionnalité dans notre cas d'étude.

6.3.2 Consommation linéaire et nombre de ressources considérées

Cette évaluation compare la résolution d'un problème de placement avec vue énergétique, selon deux modèles de consommation des serveurs. Cette consommation est dans un premier cas fonction uniquement de l'activité CPU du serveur, dans un deuxième des activités CPU et réseau du serveur.

Pour cela, nous modélisons un centre composé de deux modèles de serveurs et de deux modèles de VM, dans lequel les charges CPU, mémoire et réseau sont faibles. Nous demandons à OPTIPLACE de déterminer une configuration virtuelle de ce centre, grâce à une heuristique de réduction des coûts électriques. Cette recherche, qui s'arrête à la première solution trouvée, est effectuée de deux manières : tout d'abord en ne prenant en compte que les ressources mémoire et CPU, puis en prenant en compte les activités réseau.

Le modèle de serveur de type N1 a un CPU de 4 cœur, chacun cadencé à 2 GHz, tandis que les 16 cœurs CPU du modèle N2 sont cadencés à 3 GHz. La capacité mémoire de N1 est de 32 GB, celle de N2 de 128 GB, tandis que leur capacité de transfert réseau est respectivement de 2 et 4 GB/s. Les VM quand à elles utilisent un cœur CPU, cadencé à 50 MHz, requièrent une mémoire de 3GB pour le premier modèle et 5 GB pour le second, et écrivent sur le réseau à hauteur de 200MB/s.

Les consommations des serveurs sont linéaires avec leurs charges CPU et réseau.

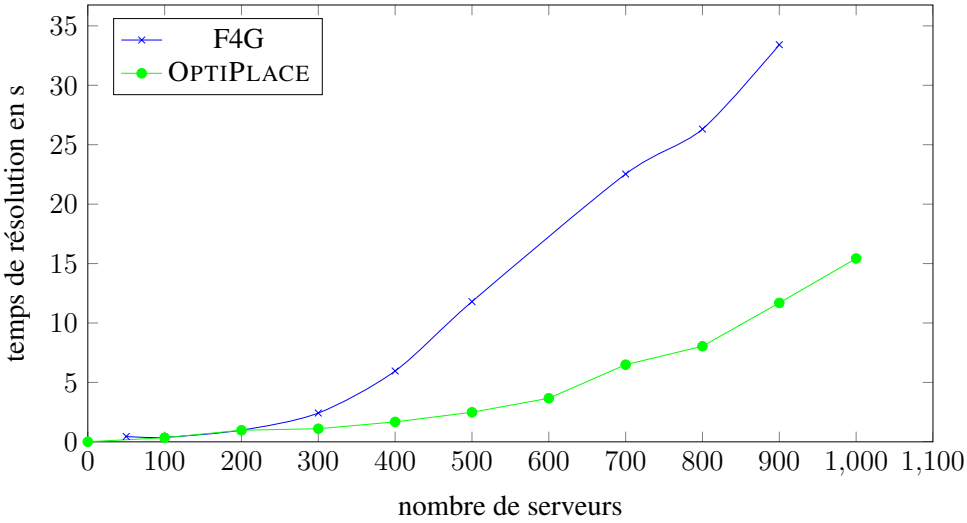


FIGURE 6.2 – Comparaison des temps de résolution pour F4G et OPTIPLACE.

La consommation minimale d'un serveur N1 est de 40W tandis que celle d'un serveur N2 est de 220W. Une augmentation de la charge CPU de 1GHz provoque une hausse de consommation de 10W pour N1, et de 7W pour N2. Enfin, une augmentation de l'activité réseau de 1GB/s provoque une hausse de consommation de 10W pour ces deux modèles.

L'unité de taille du centre est le cluster, contenant 1 serveur de type N2 et trois serveurs de type N1, ainsi que 10VM du premier modèle et 10 VM du second.

Les temps de recherche de première solution sont présentés dans la figure 6.3, en fonction du nombre de VM présentes dans le centre. Cette figure montre que si l'utilisation de plusieurs ressources dans les modèles de consommation augmente bien le temps de résolution de ce problème, cet accroissement de temps reste dans les limites du passage à l'échelle. En effet, malgré le passage d'un modèle optimisé en CPU à un modèle générique utilisant deux fois plus de ressources, les temps de résolution ont au maximum doublé.

6.3.3 Modèle énergétique avec contraintes de placement

Dans cette évaluation, nous étudions l'ajout de règles supplémentaires, le "spread" et le "gather", lors de la résolution d'un problème de placement prenant en compte les consommations de serveurs.

La règle "spread" demande à placer des VM chacune sur un serveur différent. Elle permet d'exprimer une notion de redondance parmi des VM. En effet, afin d'assurer une certaine qualité de service, des clients peuvent exécuter ces services sur plusieurs VM clones. L'utilisation de ces clones réduit la probabilité de défaut de service lorsqu'une

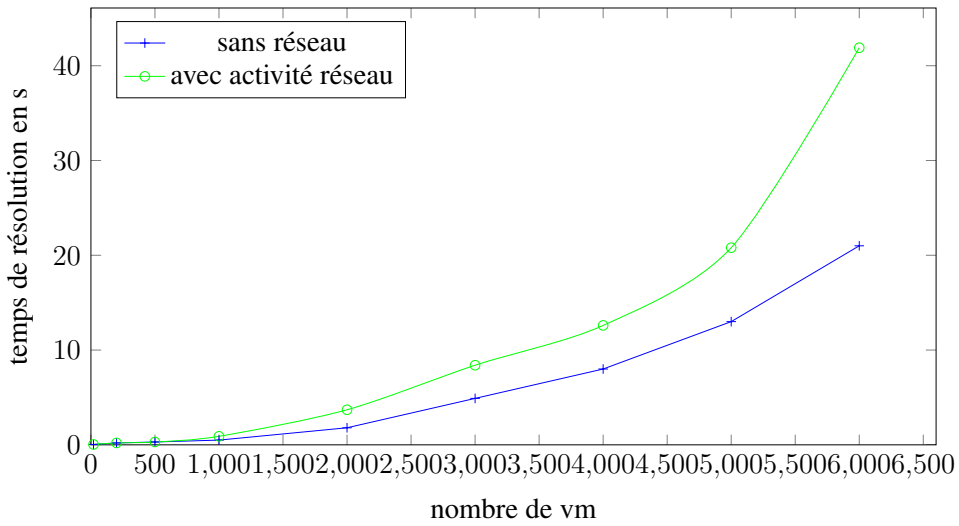


FIGURE 6.3 – temps de recherche d'une première solution selon le nombre de ressources considérées.

panne matériel contraint un serveur à s'arrêter, le service étant alors assuré par les VM clones. L'intégration de cette règle "spread" assure que les VM clonées sont toutes sur des serveurs différents, et ne peuvent donc pas être éteinte si un seul serveur tombe en panne.

La règle "gather" demande quant à elle de placer un ensemble de VM sur un même serveur. En effet, certaines VM peuvent dialoguer entre elles de manière importante, il est donc préférable de placer ces VM sur le même serveur pour réduire les temps de latence.

Ces règles sont ajoutées aux VM du centre en décomposant ce dernier en "cluster" de serveurs et VM. Un cluster est une unité de taille du centre, composé d'un nombre fixé de serveurs et de VM. Cette notion n'est utilisée que pour la définition des éléments du centre et l'ajout de règles, lors de la résolution du problème le solveur n'a aucune connaissance de ces clusters.

Chaque cluster du centre est constitué d'un serveur de modèle M1 et de 3 serveurs de modèle M2. Le modèle M1 possède une capacité de 360000CPU et 256000RAM, sa consommation variant linéairement de 520W à 1800W ; Le modèle M2 possède une capacité de 8000CPU et 48000RAM, sa consommation variant de 80W à 4000W. Ces serveurs hébergent deux modèles de VM : le modèle V1 utilisant 150CPU, le modèle V2 utilisant 200CPU, chacun utilisant aussi 4000RAM. Les serveurs de modèle M1 hébergent chacun une VM de chaque modèle, ceux de modèle M2 hébergent 8 VM de chaque modèle.

Nous évaluons le temps nécessaire à la résolution du problème de placement en ajoutant pour chaque cluster une ou plusieurs règles :

1. Pas de règle
2. Un “spread” des VM d’un M1
3. Un “spread” des VM du M2
4. Un “gather” des VM d’un M1
5. Un “gather” des VM d’un M1 et un ‘spread” des VM d’un autre M1

Les résultats, présentés dans la figure 6.4, montrent que l’ajout de ces contraintes produit un impact très faible sur le temps de recherche d’une première solution.

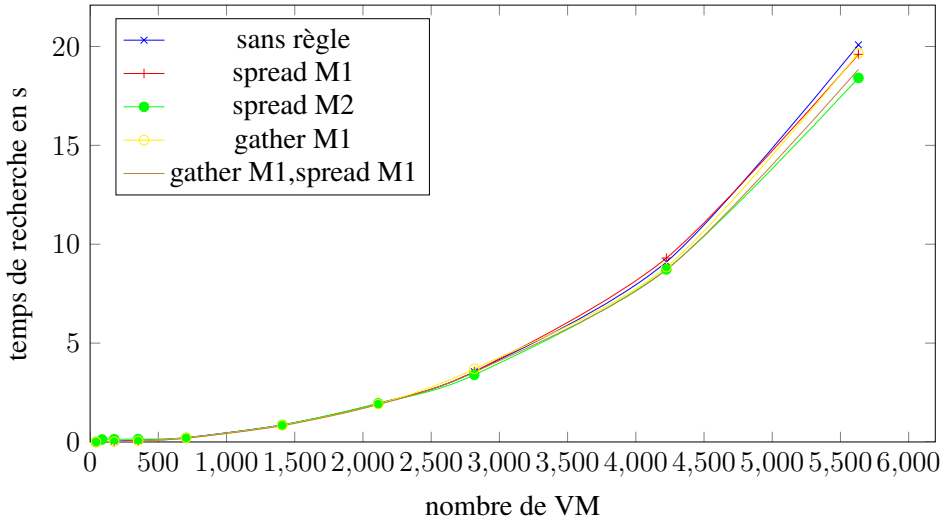
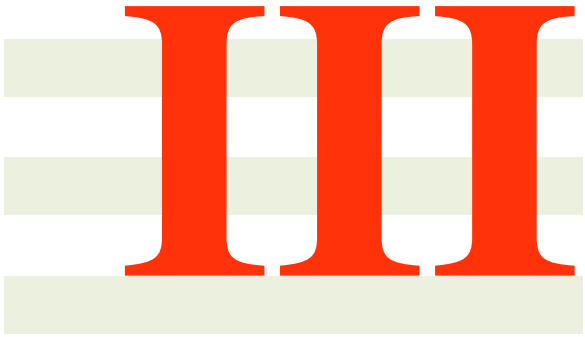


FIGURE 6.4 – Temps de recherche par nombre de clusters, pour différentes règles.

6.4 Conclusion

Dans ce chapitre nous avons évalué d’une part la capacité descriptive d’OPTIPLACE, c’est-à-dire l’intégration de modèles externes dans notre architecture, d’autre part les performances d’OPTIPLACE, que ce soit par rapport à ENTROPY ou, dans le cadre de la vue énergétique, par rapport aux travaux de F4G.



Conclusion

7

Conclusion et travaux futurs

Dans ce chapitre, nous présentons les résultats de nos travaux, puis nous proposons des améliorations dans la continuité de ces travaux.

7.1 Apport de cette thèse

Cette thèse se place dans le contexte des centres de données, des salles dédiées à l'exécution de services informatiques. Plus particulièrement, nous considérons des centres virtualisés, où les services sont encapsulés dans des environnements virtuels (ou VM) eux-même exécutés sur les serveurs.

Ces centres sont à la base du cloud computing, une tendance de l'informatique tant industrielle que scientifique, pour laquelle les infrastructures physiques, les plate-formes de service et finalement l'exécution des services clients sont proposés comme des services, utilisant des ressources extensibles et facturables à la demande. Ce modèle est dorénavant utilisé par les grandes entreprises mondiales, tant pour des plate-formes de type webserver répondant à de fréquentes requêtes, et pour lesquelles les activités des serveurs varient avec le nombre de clients accédant aux services, que pour des plate-formes de type HPC, dédiées à la résolution de problèmes complexes et nécessitant des serveurs à forte capacité de calcul et des infrastructures à haute performance.

Du fait de l'importance de ce modèle de cloud computing, le nombre de centres virtualisés et le nombre de serveurs virtualisés dans le monde n'a cessé de croître ces dernières années. Bien évidemment, la consommation électrique de ces centres a aussi connu une croissance importante. En effet, si l'efficacité électrique des serveurs a augmenté grâce aux progrès technologiques, l'accroissement du nombre de circuits par ser-

veur et donc des capacités de calcul des serveurs, d'autant plus forte dans le cas des serveurs blade, a entraîné un accroissement de leurs besoins en énergie.

Ces centres très énergivores nécessitent, pour un fonctionnement efficace, la capacité à réagir rapidement aux demandes clients, par exemple en démarrant de nouvelles VM, ou quand cela est possible en migrant les VM des serveurs surchargés vers des serveurs moins actifs. Dans les centres de grande taille, cette réaction nécessite des systèmes de gestion complexes, prenant en compte les spécificités du centre et permettant d'une part des prises de décision et des actions rapides, et d'autre part une analyse du centre et des propositions d'amélioration de celui-ci.

Le développement de tels gestionnaires soulève plusieurs problèmes. Ils doivent tout d'abord être modulables, pour permettre l'intégration de nouvelles problématiques, telle la réduction de l'empreinte énergétique du centre, la gestion des cartes réseaux ou des cartes graphiques, voire la prise en compte des contraintes thermiques du centre pour réduire les risques de dis-fonctionnement. Ils doivent aussi être validés et comparés sur des centres réels, car si l'utilisation d'un simulateur de centre permet d'obtenir des résultats, elle est limitée par la capacité des développeurs à modéliser des centres qui peuvent être très complexes.

Nous avons développé dans cette thèse deux outils. Le premier est OPTIPLACE, un système d'aide à la gestion de centre virtualisé modulaire, le deuxième est STRESSCLOUD, un système d'injection d'un scénario de charge reproductible dans un centre virtualisé.

7.1.1 OPTIPLACE

L'architecture que nous avons développée pour OPTIPLACE considère un problème de placement des VM dans un centre virtualisé. Elle permet de prendre en compte des problèmes variés, sans devoir modifier le cœur de métier, par l'utilisation d'un système de vues dédiées chacune à la représentation et la résolution d'un problème. De plus, l'accès aux composants internes d'OPTIPLACE permet de développer des stratégies de recherche de solutions adaptées à des problèmes de grande taille. Cette modularité permet aussi le passage à l'échelle par la désactivation de certains modules, permettant de réduire le temps nécessaire pour déterminer les solutions au problème. Finalement, la séparation du problème en vues permet aux développeurs de travailler sur des sous-problèmes différents, puis de mettre en commun les résultats de leurs travaux dans OPTIPLACE.

En utilisant cette architecture, nous avons développé un module énergétique associant aux serveurs du centre des modèles de consommation électrique. Ce module permet de réduire la consommation du centre et contient des heuristiques pour améliorer le temps de recherche dans ce sens. Il propose aussi des contraintes énergétiques pour par exemple limiter la consommation de certains serveurs ou groupes de serveurs. Les modèles énergétiques proposés sont modulaires et modifiables par des développeurs. Ils

sont intégrés dans un problème de manière explicite, et peuvent donc être utilisés par d'autres vues du problème dans des travaux ultérieurs (par exemple, une vue thermique).

7.1.2 STRESSCLOUD

Les travaux de gestion d'un centre nécessitent l'injection de charges système de manière reproductible et scénarisée. Afin de satisfaire ce besoin, nous avons développé STRESSCLOUD, un outil d'injection de charge distribuée dans un centre de type IaaS. Cet outil utilise un langage dédié, basé sur Groovy, pour sélectionner les VM du centre et leur injecter un scénario d'activités. Il nous est ainsi possible de jouer le même scénario d'activités plusieurs fois, tout en s'assurant que les VM auront les mêmes injections d'activité à chaque exécution. Après chaque exécution d'un scénario, STRESSCLOUD permet d'obtenir les données de performances des VM lors de cette exécution.

Outre l'évaluation des gestionnaires de centre, cet outil nous permet de paramétrer des modèles de comportement du centre en fonction des activités des VM. Nous pouvons par exemple, après avoir installé des sondes énergétiques, injecter différentes charges dans les VM et en déduire des modèles de consommation des serveurs. Cet outil nous permet donc d'une part de modéliser le comportement d'un centre dans OPTIPLACE et d'autre part d'évaluer l'action d'un SGIV sur ce centre.

Ces deux outils ont été développés dans un cadre spécifique, ils peuvent être adaptés (ou, pour OPTIPLACE, enrichis) pour correspondre à des besoins différents, que nous détaillons dans la section suivante des travaux futurs.

7.2 Travaux futurs

Cette section décrit des modifications qui peuvent être apportées aux outils développés, mais qui nécessitaient un travail trop important ou dépassaient le cadre de la thèse.

7.2.1 Intégration de OPTIPLACE dans BTRCLOUD

OPTIPLACE a été développé pour intégrer différents paramètres d'un centre virtualisé dans la résolution d'un problème de placement de VM en tant que brique logicielle de BTRCLOUD. Nous ne nous sommes actuellement préoccupé que de la résolution de problèmes, il nous faut donc développer un pont entre BTRCLOUD et OPTIPLACE.

Ce pont doit permettre le dialogue entre BTRCLOUD et OPTIPLACE, et donc non seulement l'analyse du centre observé par BTRCLOUD et la déduction d'actions d'administration à la demande, mais aussi la présentation du système à un administrateur de centre, via une interface graphique dédiée. En particulier, même si l'administrateur n'a pas connaissance de l'architecture en vues, les notions de contraintes dans le centre,

d'objectif de résolution, de spécification des ressources du centre nécessitent un soin particulier de présentation.

Ces notions sont présentes dans OPTIPLACE pour permettre à un développeur de les adapter à une problématique dans une vue, et ainsi de modifier leur sens. Cependant, pour présenter ce nouveau sens à un administrateur, il est nécessaire d'enrichir OPTIPLACE d'un système de plugin qui permette à OPTIPLACE de découvrir et de charger dynamiquement les vues proposées par les développeurs et de comprendre cette nouvelle sémantique. Par exemple, une vue intégrant des notions de réservation des ressources à un problème de placement doit indiquer à l'administrateur en quoi le placement actuel d'un centre n'est pas correct, lorsque c'est le cas.

Enfin, certaines vues utilisent des modèles issus des observations du centres, par exemple les consommations des serveurs peuvent être déduites d'observations des wattmètres et de l'activité des serveurs. L'observation du centre étant une préoccupation de btrMonitor, il peut être intéressant de lier ces vues à des modules de BTRCLOUD. De même que pour l'intégration des vues dans OPTIPLACE ou pour l'interaction de ces vues avec l'utilisateur, l'interaction d'OPTIPLACE avec BTRCLOUD nécessite des travaux poussés d'architecture.

Outre ces travaux d'architecture, il nous parait intéressant de modéliser les problèmes d'extinction des serveurs dans le centre.

7.2.2 Modèle d'extinction des serveurs

Actuellement, OPTIPLACE ne maîtrise pas la notion d'allumage, ou d'extinction, des serveurs lors d'une reconfiguration. La vue énergétique peut considérer qu'un serveur sans VM est éteint, et a donc une consommation nulle, cependant ce modèle demande à être affiné.

En effet, des contraintes spécifiques à l'allumage et l'extinction des serveurs sont intéressantes pour permettre le provisionnement des serveurs, par exemple en demandant d'avoir toujours 3 serveurs inactifs pour y démarrer les nouvelles VM. Si de telles contraintes pourraient être intégrées par l'ajout de fausses VM à provisionner, cette modification du centre aurait des effets néfastes sur l'intégration d'autres vues, par exemple ces fausses VM pourraient être migrées ou participer à la consommation des serveurs dans la vue énergétique. De fait, les notions de provisionnement et d'extinction des serveurs sont plus complexes à intégrer dans un problème de PPC qu'il n'y parait.

Ces notions nécessiteraient l'intégration d'une sémantique d'allumage des serveurs dans OPTIPLACE, et des règles associées. D'une part, un administrateur doit pouvoir demander à conserver des serveurs inutilisés pour amortir une hausse d'activité du centre. D'autre part, ces notions permettent de décider quels serveurs éteindre ou allumer, et donc d'affiner le modèle de consommation du centre.

7.2.3 Planification des tâches et actions

La version 2.0 d'ENTROPY comportait un système de planification des actions d'administration, que nous n'avons pas intégré dans OPTIPLACE car il ne concernait pas notre problème de réduction de l'empreinte énergétique. Pour réintégrer les travaux effectués par Fabien Hermenier, il faudrait donc développer un module dédié à la définition des actions d'administration et leur planification dans OPTIPLACE.

En plus de la planification des actions, la planification temporelle des VM de type HPC est une propriété qui pourrait faire l'objet d'un module d'OPTIPLACE. En effet, dans les centres de type HPC, certaines tâches telles la sauvegarde d'une base de données ou l'analyse de données scientifiques peuvent être exécutées à des moments variables. Le démarrage de ces tâches peut être planifiée pour prendre en compte différents paramètres du centre, telle la consommation des serveurs. Il peut donc être intéressant de développer un module de planification temporelle des tâches, pour intégrer de tels travaux, après avoir développé un module de planification des actions qui intègre la notion d'action d'administration.

Enfin, ces notions de sémantique et d'actions intégrées à OPTIPLACE nous pourrons nous concentrer sur la modélisation des problèmes thermiques dans un centre.

7.2.4 Impacts thermiques linéaires

L'objectif initial de cette thèse était l'intégration non seulement de modèles énergétiques dans ENTROPY, mais aussi celle de modèles des flux de chaleur. En particulier, nous voulions intégrer les travaux de l'Impact Lab [VBG09], qui adressent des problèmes de réduction des points chauds et d'optimisation des systèmes de refroidissement.

Problématique de recirculation de l'air

Dans ces travaux, cette équipe considère un centre virtualisé dont les serveurs sont refroidis par l'air ambiant refroidit par un système de climatisation. Dans ceux-ci, un phénomène de recirculation de l'air réchauffe les serveurs du centre et réduit l'efficacité de la climatisation. En effet, l'air qui sort de cette climatisation est réchauffée par l'activité du centre avant d'atteindre un serveur à refroidir (typiquement, en façade du serveur), du fait du mélange de l'air froid de la climatisation avec l'air chaud en sortie des serveurs. Le système de refroidissement doit donc adapter la température de l'air pour éviter que ce réchauffement local n'endommage le matériel. Dans les centres traditionnels, ce problème est compensé par une marge de température en sortie de la climatisation, nécessitant donc un travail accru de refroidissement de l'air. Cette équipe propose donc un placement des VM optimisé pour réduire le besoin de refroidissement de l'air.

Une modélisation complète de ce phénomène serait très coûteuse en terme de temps de calcul, son intégration dans un système tel que OPTIPLACE impossible. En effet, les modèles de dynamique des fluides nécessaires à la prise en compte d'un centre nécessitent un travail de description du centre et des temps de résolution de l'ordre de l'heure pour des centres de taille moyenne.

Ces travaux considèrent différents types de centre virtualisé, ainsi que différents modèles de climatisation, et associent à chaque centre des fonctions d'impacts thermiques entre les serveurs. Ils considèrent que la fonction d'accroissement de température en entrée d'un serveur est linéaire avec les consommations électriques des serveurs du centre. Ainsi, lorsque les consommations des serveurs sont linéaires avec leur activité CPU, la fonction d'accroissement de température d'un serveur est linéaire avec les activités CPU des serveurs du centre. Ce modèle est donc intégrable par une simple matrice d'impact linéaire entre les serveurs, modélisant la propension de chaque serveur à échauffer les autres. Ce modèle dit d'*impacts thermiques linéaires* permet d'intégrer facilement cette notion de recirculation lors de la résolution d'un problème de placement.

Ainssi, un problème de placement utilisant ce modèle linéaire peut être résolu facilement avec la programmation linéaire. Ce modèle est aussi intégrable simplement dans un solveur de PPC, ce qui permet de comparer différentes approches. Ce modèle simplifié d'impacts linéaires permet donc d'intégrer des modèles complexes de dynamique des fluides dans des problèmes qui ne s'y prêtent pas. Il demande cependant une phase de paramétrage pour être utilisé.

Déduction des impacts dans des centres génériques

Un tel modèle demande un nombre réduit d'observations pour être paramétré En effet, dans un centre comportant n serveurs, le modèle d'impacts linéaire associe à chacun de ces serveur une équation d'augmentation de la température. L'équation de chaque serveur étant linéaire avec les consommations des autres serveurs, elle est paramétrée par n coefficients d'impact, et peut donc être déduite à partir de n observations des consommations et températures du centre suffisamment différentes. Si chaque serveur met à disposition une sonde thermique en façade et une sonde énergétique, et si les activités (et donc consommations) des serveurs varient suffisamment, ce modèle peut être déduit automatiquement à partir d'observations d'une activité réelle du centre, ce qui limite les besoins d'interaction d'un administrateur.

Les valeurs de consommation des serveurs sont généralement accessibles dans un centre virtualisé. En effet, la plupart des constructeurs de serveur met à disposition des sondes énergétiques, et souvent thermiques, intégrées par exemple dans une carte de gestion physique. Lorsque ce n'est pas le cas, il peut être nécessaire de placer des sondes énergétiques sur les alimentations des serveurs. Afin d'obtenir des plages d'observation suffisamment larges, il peut être intéressant de faire varier les activités des serveurs, par exemple avec un injecteur de charge tel que STRESSCLOUD ou par la migration des

VM d'un serveur. À condition d'avoir les droits suffisants, l'obtention de ces valeurs de consommation n'est donc généralement pas un problème.

L'observation des températures en entrée des serveurs, par contre, n'est pas aussi évidente. Si les constructeurs mettent à disposition es sondes thermiques dans les serveurs, celles-ci sont généralement placées à l'intérieur de ceux-ci, par exemple dans le CPU ou la carte mère. Il n'est donc pas possible d'interroger directement les serveurs comme c'est le cas pour leur consommation.

Si des techniques permettent, par exemple avec des caméras infrarouge [HLT12], de connaître les températures en différents points d'une salle, leur utilisation dans un centre de grande taille n'est pas chose aisée et peut nécessiter des contraintes trop importantes, telle que l'arrêt des activités des serveurs.

Les travaux de l'Impact Lab on porté sur des centres de moins d'une dizaine de serveurs. Pour évaluer leurs travaux, ils ont utilisé un simulateur de fluide thermique, pour déterminer les valeurs de température de l'air en entrée des serveurs selon la charge électrique des serveurs. Les travaux de l'Impact Lab ont de plus intégré des modèles d'efficacité du système de refroidissement.

Simulation thermique d'un centre

La simulation thermique d'un centre consiste à déterminer, en fonction l'activité énergétique des éléments du centre, ici des serveurs, la température, la vitesse et la pression de l'air en différents points du centre. Pour cela, elle nécessite de décrire l'architecture physique ainsi que les matériaux du centre, de déterminer les valeurs de consommation électrique des serveurs, et enfin d'indiquer les conditions de l'air produite par la climatisation. Cette simulation thermique permet d'une part de déterminer les problèmes d'architecture physique du centre, telle la présence de zones non refroidis, pouvant produire des points chauds, mais aussi d'avoir accès aux informations de température en entrée des serveurs.

Des outils de modélisation thermiques sont disponibles dans cette optique. De nombreux outils commerciaux permettent de modéliser, à partir d'une définition physique du centre, d'une bibliothèque des propriété physiques des matériaux, et des données de réchauffement et refroidissement dans le centre, les flux d'air du centre dans un état d'équilibre. Ces outils utilisent chacun un solveur de CFD, qui découpe l'espace du centre en zones et considère des équations entre les valeurs de pression, température, vitesses de ces zones, selon les matériaux qui constituent celles-ci. Ce solveur effectue ensuite des opérations de propagation de ces équations dans les variables des zones, convergeant vers un point d'équilibre, ne pouvant être amélioré, considéré comme la solution du problème de modélisation. Le temps nécessaire à cette résolution est d'autant plus important que le nombre d'éléments à considérer est important. Ce nombre augmente bien évidemment avec le nombre de serveurs et l'espace occupé par le centre, mais aussi avec la précision du modèle, c'est à dire le nombre de zones dans lesquelles

le centre a été décomposé. De plus, la définition du centre, c'est-à-dire la description des éléments physiques qui le composent, demande aussi un temps proportionnel au nombre de ces éléments.

Cependant, si des outils commerciaux permettent de définir un centre et de modéliser ses flux thermiques, ceux que nous avons évalués agissent tous uniquement via une interface graphique. Or nous voulons utiliser de telles modélisations pour paramétrer des équations comportant un nombre important de variables, de l'ordre de la centaine. Il est alors nécessaire, pour générer une centaine de simulations, de réaliser une centaine de modifications manuelles du problème considéré, avant de relancer à chaque fois une simulation. Ce travail fastidieux augmente grandement la probabilité de générer des erreurs, nous considérons pour l'instant impossible de modéliser efficacement les flux thermiques pour une centaine de paramétrage des consommations des serveurs.

Ainsi, il n'est pour l'instant pas possible de modéliser les impacts linéaires entre les serveurs sur des centres de grande taille. Pour intégrer de tels travaux dans OPTIPLACE, nous nécessitons un système de simulation des fluides en mode automatique, si possible réparti pour réduire le temps nécessaire à la résolution de l'ensemble du problème, que nous n'avons pu trouver.

7.2.5 Évolution de STRESSCLOUD

Bien qu'il satisfasse les besoins qui ont conduit à sa création, STRESSCLOUD peut être amélioré. D'une part, il n'est pas encore adapté à des centre de très grande capacité. D'autre part, il peut être utilisé dans des cadres plus large que ceux prévus initialement, en particulier certains cas nécessitent une liaison au SGIV.

Centre de grande capacité

STRESSCLOUD a été développé avec des objectifs fonctionnels et testé sur un centre de petite taille. Les performances du serveur central n'étaient alors pas une priorité de développement. Il convient, pour permettre son utilisation dans des centres de très grande taille, d'assurer ces performances en adaptant non seulement le code, mais aussi l'architecture de STRESSCLOUD à une telle utilisation.

Premièrement, le code peut être optimisé pour un passage à l'échelle. Par exemple, la gestion du réseau utilise des classes et une architecture choisies pour leur simplicité d'utilisation. Une optimisation du code peut être faite en utilisant des classes développées pour accepter une montée en charge, par exemple celles du package `java.nio` en Java. De même, les codes de recherche, de parcours, de mémorisation des informations pourraient bénéficier d'adaptations à une montée en charge du système, telles des systèmes de cache des requêtes.

Ensuite, lorsque le nombre de VM présentes augmente, le serveur central a plus de difficultés à les manipuler, de par la nécessité de parcourir des boucles algorithmiques

sur des ensembles de grande taille. Le regroupement des VM contrôlées permettrait de réduire le nombre d'objets à manipuler par le serveur d'enregistrement. En effet, lors de l'utilisation d'un nombre de VM important dans un scénario, plusieurs VM auront exactement les mêmes demandes de charge. Il est donc intéressant d'exprimer dans le langage des scénario cette sémantique.

Cette notion permettait de réduire la taille des scénarios, mais aussi la charge réseau du serveur d'enregistrement. En effet, cette notion de groupe permet d'intégrer dans la couche réseau la diffusion de groupe. Ainsi, une commande ne serait plus émise une fois pour chaque VM du groupe, mais une seule fois vers le groupe. Cette optimisation demande cependant un déploiement spécifique pour être effective, nécessitant un travail d'adaptation de l'architecture réseau.

Finalement, l'architecture réseau peut profiter de ces améliorations. En effet, si actuellement chaque VM se connecte au registar, celui-ci doit donc maintenir autant de connections réseau qu'il y a de VM configurées pour STRESSCLOUD. Or, les performances d'un serveur réseau décroissent avec le nombre de connections que celui-ci doit maintenir, puisqu'à même capacité de calcul, il doit servir plus de requêtes. Pour améliorer ces performances, nous proposons une architecture réseau répartie en arbre, où la racine est le registar, où les feuilles sont les VM, et où chaque nœud intermédiaire correspond à un serveur relais, mutualisant les connections de ses fils vers son père. En plus de réduire le nombre de connections au registar, cette architecture permettrait d'implémenter la notion de groupes de VM au niveau du réseau, présentée plus haut.

Ces modifications concernent la montée en charge de STRESSCLOUD, cependant les fonctionnalités de STRESSCLOUD peuvent aussi être revues.

Liaison au SGIV

Les problèmes principaux lors de l'utilisation de STRESSCLOUD sont, d'une part, l'assurance d'avoir un nombre de VM suffisant pour exécuter le scénario et, d'autre part, d'assurer un même état initial du centre lors de l'exécution de plusieurs scénarios successifs.

Le premier problème ne peut être résolu que par l'action d'un administrateur du centre. En effet il n'est pas possible de créer des VM sans connaissance des besoins du scénario, par exemple de la mémoire qu'il faut attribuer à chaque VM, ou du nombre de cœurs CPU. Bien qu'il soit possible de créer un modèle de VM, puis de le cloner pour créer une VM participante, cette fonctionnalité demande généralement une quantité importante d'actions de la part de l'administrateur. Ce problème de création de VM, qui peut être coûteux pour l'administrateur en terme d'actions à réaliser, doit donc être résolu en dehors de STRESSCLOUD.

Le problème de maintient de l'état initial du centre, quant à lui, ne peut être facilement corrigé par un administrateur seul. Ce problème consiste à assurer que lors de l'exécution du même scénario d'activité deux fois, le positionnement des VM sur les

serveurs et les états de fonctionnement des serveurs seront les mêmes. En effet, lors de la comparaison de stratégies de gestion de centre, il est nécessaire d'exécuter plusieurs fois le même scénario, et donc de s'assurer que toutes les VM du centre sont dans le même état initial. Si les centres de petite taille peuvent être manuellement réorganisés par l'administrateur, cette gestion devient problématique lors de l'accroissement de la taille du centre. Il devient donc intéressant de permettre dans STRESSCLOUD de réinitialiser la configuration virtuelle du centre.

Idéalement, les scénarios de STRESSCLOUD seraient invoqués directement par le SGIV du centre, qui se chargerait donc de reconfigurer ce centre avant d'exécuter un scénario. Cependant, bien que STRESSCLOUD dispose d'une interface REST, il n'est pas possible de modifier les SGIV pour y intégrer les accès à STRESSCLOUD. Si certains SGIV permettent l'exécution de scripts, et que les scénarios de STRESSCLOUD peuvent être controlés à distance, cette fonctionnalité ne peut être toujours assurée.

Il est alors utile d'avoir un accès à certaines fonctionnalités du SGIV dans STRESSCLOUD, par exemple la restitution de l'état d'un centre dans un état initial. Un utilisateur de STRESSCLOUD pourrait alors demander la mémorisation de l'état du centre et restituer cet état après chaque scénario.

Si l'intégration de l'accès au SGIV dans STRESSCLOUD semble être une bonne idée, en pratique, tel que vu avec ENTROPY, cette intégration demande le développement et le maintient d'un nombre important de drivers. Or, BTRPLACE intègre justement de tels drivers et se focalise sur l'observation d'un centre virtualisé. L'intégration d'un pont vers BTRPLACE dans STRESSCLOUD permettrait donc de réduire le coût de cette maintenance, qui est déjà prise en charge par BTRPLACE. De plus, une telle liaison permettrait d'évaluer plus finement les performances des algorithmes de gestion du centre, puisque donnant accès à des valeurs de performance disponibles dans les SGIV.

Liste des tableaux

Table des figures

Bibliographie

[AFG+10] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, et al. A view of cloud computing. Communications of the ACM, 53(4) :50–58, 2010.

[AMVG11] Zahra Abbasi, Tridib Mukherjee, Georgios Varsamopoulos, and Sandeep K. S. Gupta. Dynamic hosting management of web based applications over clouds. In International Conference on High performance Computing Conference (HiPC2011), dec 2011.

[ASR+10] S. Akoush, R. Sohan, A. Rice, A.W. Moore, and A. Hopper. Predicting the performance of virtual machine migration. In Modeling, Analysis Simulation of Computer and Telecommunication Systems (MASCOTS), 2010 IEEE International Symposium on, pages 37–46, 2010.

[AT09] E. Ayguadé and J. Torres. Creating power-aware middleware for energy-efficient data centers. In ERCIM News, 79, oct 2009.

[Bar05] Luiz André Barroso. The price of performance. Queue, 3 :48–53, September 2005.

[BBG+10] Robert Basmadjian, Christian Bunse, Vasiliki Georgiadou, Giovanni Giuliani, Sonja Klingert, Gergö Lovasz, and Mikko Majanen. Fit4green-energy aware ict optimization policies. Proceedings of the COST Action IC0804 on Energy Efficiency in Large Scale Distributed Systems-1st Year, 2010.

[Bel00] Frank Bellosa. The benefits of event : driven energy accounting in power-sensitive systems. In Proceedings of the 9th workshop on ACM SIGOPS European workshop : beyond the PC : new challenges for the operating system, pages 37–42. ACM, 2000.

[BF07] Cullen Bash and George Forman. Cool job allocation : Measuring the power savings of placing jobs at cooling-efficient locations in the data center. In USENIX Annual Technical Conference, volume 138, page 140, 2007.

[BGDG+10] Andreas Berl, Erol Gelenbe, Marco Di Girolamo, Giovanni Giuliani, Hermann De Meer, Minh Quan Dang, and Kostas Pentikousis. Energy-efficient cloud computing. The Computer Journal, 53(7) :1045–1051, 2010.

[BH07] Luiz André Barroso and Urs Holzle. The case for energy-proportional computing. Computer, 40(12) :33–37, 2007.

[BM07] Christian L Belady and Christopher G Malone. Metrics and an infrastructure model to evaluate data center efficiency. In ASME 2007 InterPACK Conference collocated with the ASME/JSME 2007 Thermal Engineering Heat Transfer Summer Conference, pages 751–755. American Society of Mechanical Engineers, 2007.

[BS06] Kenneth A Barclay and W. J Savage. Groovy programming. Morgan Kaufmann Publishers, San Francisco, CA, 2006.

[BS07] Belaıd Benhamou and Mohamed Réda Saıdi. Dynamic detection and elimination of local symmetry in csps. In The CP 2007 Workshop on Symmetry and Constraint Satisfaction Problems (SymCon'07). Providence, USA, pages 22–29, 2007.

[BYV+09] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. Cloud computing and emerging platforms : Vision, hype, and reality for delivering computing as the 5th utility. Future Generation Computer Systems, 25(6) :599 – 616, 2009.

[CADAD+97] David E Culler, Andrea Arpaci-Dusseau, Remzi Arpaci-Dusseau, Brent Chun, Steven Lumetta, Alan Mainwaring, Richard Martin, Chad Yoshikawa, and Frederick Wong. Parallel computing on the berkeley now. In 9th Joint Symposium on Parallel Processing, 1997.

[cho] choco. choco solver. http://www.emn.fr/z-info/choco-solver/.

[CKS09] Michael Cardosa, Madhukar R Korupolu, and Aameek Singh. Shares and utilities based power consolidation in virtualized server environments. In Integrated Network Management, 2009. IM'09. IFIP/IEEE International Symposium on, pages 327–334. IEEE, 2009.

[Col08] J-P Colinge. FinFETs and other multi-gate transistors. Springer, 2008.

[DHKC09] Stephen Dawson-Haggerty, Andrew Krioukov, and David E. Culler. Power optimization, a reality check. Technical Report UCB/EECS-2009-140, EECS Department, University of California, Berkeley, Oct 2009.

[Dil08] Bruno Dillenseger. CLIF, a framework based on fractal for flexible, distributed load testing. annals of telecommunications - annales des télécommunications, 64(1-2) :101–120, November 2008.

[DMR10] Gaurav Dhiman, Giacomo Marchetti, and Tajana Rosing. vgreen : A system for energy-efficient management of virtual machines. ACM Trans. Des. Autom. Electron. Syst., 16(1) :6 :1–6 :27, November 2010.

[FDF03] Renato J Figueiredo, Peter A Dinda, and José AB Fortes. A case for grid computing on virtual machines. In Distributed Computing Systems, 2003. Proceedings. 23rd International Conference on, pages 550–559. IEEE, 2003.

[FRM12] Eugen Feller, Louis Rilling, and Christine Morin. Snooze : A scalable and autonomic virtual machine management framework for private clouds. In Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012), pages 482–489. IEEE Computer Society, 2012.

[FVdW02] Michael Frumkin and Rob F. Van der Wijngaart. Nas grid benchmarks : A tool for grid space exploration. Cluster Computing, 5(3) :247–255, July 2002.

[Gar07] Simson L Garfinkel. An evaluation of amazon's grid computing services : Ec2, s3, and sqs. In Center for. Citeseer, 2007.

[GHBK11] Anshul Gandhi, Mor Harchol-Balter, and Michael A Kozuch. The case for sleep states in servers. In Proceedings of the 4th Workshop on Power-Aware Computing and Systems, page 2. ACM, 2011.

[GHBK12] Anshul Gandhi, Mor Harchol-Balter, and Michael A Kozuch. Are sleep states effective in data centers ? In Green Computing Conference (IGCC), 2012 International, pages 1–10. IEEE, 2012.

[GL10] Jérôme Gallard and Adrien Lèbre. Managing virtual resources : Fly through the sky. ERCIM News, pages 36–37, 10 2010.

[Gra09] Kathy Gray. Tport deal with google to create jobs. Technical report, The Dalles Chronicle, http ://www.gorgebusiness.com/2005/ google.htm, 2009.

[Ham11] James Hamilton. Designing efficient internet scale services part ii, june 2011.

[HAP+05] Mark Horowitz, Elad Alon, Dinesh Patil, Samuel Naffziger, Rajesh Kumar, and Kerry Bernstein. Scaling, power, and the future of cmos. In Electron Devices Meeting, 2005. IEDM Technical Digest. IEEE International, pages 7–pp. IEEE, 2005.

[Her09] Fabien Hermenier. Gestion dynamique des tâches dans les grappes, une approche à base de machines virtuelles. Thèse, Université de Nantes, November 2009.

[HF05] Chung-hsing Hsu and Wu-chun Feng. A power-aware run-time system for high-performance computing. In Proceedings of the 2005 ACM/IEEE conference on Supercomputing, page 1. IEEE Computer Society, 2005.

[Hir05] Radhakrishna Hiremane. From moore's law to intel innovation—prediction to reality. Technology, 1, 2005.

[HLAP06] Wei Huang, Jiuxing Liu, Bulent Abali, and Dhabaleswar K Panda. A case for high performance computing with virtual machines. In Proceedings of the 20th annual international conference on Supercomputing, pages 125–134. ACM, 2006.

[HLM+09] Fabien Hermenier, Xavier Lorca, Jean-Marc Menaud, Gilles Muller, and Julia Lawall. Entropy : a consolidation manager for clusters. In Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments, VEE '09, pages 41–50, New York, NY, USA, 2009. ACM.

[HLT12] Jhen-Jia Hu, Hui-Chieh Li, and Hung-Ming Tai. Thermal distribution monitoring of the container data center by a fast infrared image fusion technique. Comput. Math. Appl., 64(5) :1484–1494, September 2012.

[HM07] Sebastian Herbert and Diana Marculescu. Analysis of dynamic voltage/frequency scaling in chip-multiprocessors. In Low Power Electronics and Design (ISLPED), 2007 ACM/IEEE International Symposium on, pages 38–43. IEEE, 2007.

[IMK+13] Canturk Isci, Suzanne McIntosh, Jeffrey Kephart, Rajarshi Das, James Hanson, Scott Piper, Robert Wolford, Thomas Brey, Robert Kantner, Allen Ng, et al. Agile, efficient virtualization power management with low-latency server power states. In Proceedings of the 40th Annual International Symposium on Computer Architecture, pages 96–107. ACM, 2013.

[JHJ+10] Gueyoung Jung, Matti A. Hiltunen, Kaustubh R. Joshi, Richard D. Schlichting, and Calton Pu. Mistral : Dynamically managing power, performance, and adaptation cost in cloud infrastructures. In Proceedings of the 2010 IEEE 30th International Conference on Distributed Computing Systems, ICDCS '10, pages 62–73, Washington, DC, USA, 2010. IEEE Computer Society.

[JMSN05] Hailin Jiang, Malgorzata Marek-Sadowska, and Sani R Nassif. Benefits and costs of power-gating technique. In Computer Design : VLSI in Computers and Processors, 2005. ICCD 2005. Proceedings. 2005 IEEE International Conference on, pages 559–566. IEEE, 2005.

[K W07] K Wensel, Chris. JStress : a performance profiling harness. http ://jstress.manamplified.org/, September 2007.

[KAGS11] Bhavani Krishnan, Hrishikesh Amur, Ada Gavrilovska, and Karsten Schwan. Vm power metering : feasibility and challenges. ACM SIGMETRICS Performance Evaluation Review, 38(3) :56–60, 2011.

[Kat09] R. H. Katz. Tech titans building boom. Technical report, IEEE Spectrum, 2009.

[KBKK06] Gunjan Khanna, Kirk Beaty, Gautam Kar, and Andrzej Kochut. Application performance management in virtualized server environments. In Network Operations and Management Symposium, 2006. NOMS 2006. 10th IEEE/IFIP, pages 373–381. IEEE, 2006.

[KFJ+03] Rakesh Kumar, Keith I Farkas, Norman P Jouppi, Parthasarathy Ranganathan, and Dean M Tullsen. Single-isa heterogeneous multi-core architectures : The potential for processor power reduction. In Microarchitecture, 2003. MICRO-36. Proceedings. 36th Annual IEEE/ACM International Symposium on, pages 81–92. IEEE, 2003.

[KHS07] Ron Kohavi, Randal M Henne, and Dan Sommerfield. Practical guide to controlled experiments on the web : listen to your customers not to the hippo. In Proceedings of the 13th ACM SIGKDD international

conference on Knowledge discovery and data mining, pages 959–967. ACM, 2007.

[Koo11] Jonathan Koomey. Growth in data center electricity use 2005 to 2010. Oakland, CA : Analytics Press. August, 1 :2010, 2011.

[KPR09] Balachandra Reddy Kandukuri, V Ramakrishna Paturi, and Atanu Rakshit. Cloud security issues. In Services Computing, 2009. SCC'09. IEEE International Conference on, pages 517–520. IEEE, 2009.

[KZL+10] Aman Kansal, Feng Zhao, Jie Liu, Nupur Kothari, and Arka A. Bhattacharya. Virtual machine power metering and provisioning. In Proceedings of the 1st ACM Symposium on Cloud Computing, SoCC '10, pages 39–50, New York, NY, USA, 2010. ACM.

[Lau83] Eric Laurent. La puce et les géants. De la révolution informatique à la guerre du renseignement. Fayard, 1983.

[LLH+09] Bo Li, Jianxin Li, Jinpeng Huai, Tianyu Wo, Qin Li, and Liang Zhong. Enacloud : An energy-saving application live placement approach for cloud computing environments. In Proceedings of the 2009 IEEE International Conference on Cloud Computing, CLOUD '09, pages 17–24, Washington, DC, USA, 2009. IEEE Computer Society.

[MBV+09] Tridib Mukherjee, Ayan Banerjee, Georgios Varsamopoulos, Sandeep K. S. Gupta, and Sanjay Rungta. Spatio-temporal thermal-aware job scheduling to minimize energy consumption in virtualized heterogeneous data centers. Comput. Netw., 53 :2888–2904, December 2009.

[MCC04] Matthew L. Massie, Brent N. Chun, and David E. Culler. The ganglia distributed monitoring system : design, implementation, and experience. Parallel Computing, 30 :817–840, 2004.

[MCRS05] Justin D Moore, Jeffrey S Chase, Parthasarathy Ranganathan, and Ratnesh K Sharma. Making scheduling "cool" : Temperature-aware workload placement in data centers. In USENIX annual technical conference, General Track, pages 61–75, 2005.

[ME09] Michael Schaefer Marc Tu Duy Khiem Mike Ebbers, Alvin Galea. The green data center : Steps for the journey. Technical report, IBM, 2009.

[MGW09] David Meisner, Brian T. Gold, and Thomas F. Wenisch. Powernap : eliminating server idle power. 2009.

[MV08] John Michalakes and Manish Vachharajani. Gpu acceleration of numerical weather prediction. Parallel Processing Letters, 18(04) :531–548, 2008.

[MWY+10] Haibo Mi, Huaimin Wang, Gang Yin, Yangfan Zhou, Dianxi Shi, and Lin Yuan. Online self-reconfiguration with performance guarantee for energy-efficient large-scale cloud computing data centers. In Proceedings of the 2010 IEEE International Conference on Services Computing, SCC '10, pages 514–521, Washington, DC, USA, 2010. IEEE Computer Society.

[NIG07] Ripal Nathuji, Canturk Isci, and Eugene Gorbatov. Exploiting platform heterogeneity for power efficient data centers. In Autonomic Computing, 2007. ICAC'07. Fourth International Conference on, pages 5–5. IEEE, 2007.

[PBB+01] Chandrakant D Patel, Cullen E Bash, Christian Belady, Lennart Stahl, and Danny Sullivan. Computational fluid dynamics modeling of high compute density data centers to assure system inlet air specifications. In Proceedings of IPACK, volume 1, pages 8–13, 2001.

[Pot12] Rémy Pottier. Un langage dédié à l'administration d'infrastructures virtualisées. These, Ecole des Mines de Nantes, September 2012.

[RRK08] Suzanne Rivoire, Parthasarathy Ranganathan, and Christos Kozyrakis. A comparison of high-level full-system power models. HotPower, 8 :3–3, 2008.

[SBP+05] Ratnesh K Sharma, Cullen E Bash, Chandrakant D Patel, Richard J Friedrich, and Jeffrey S Chase. Balance of power : Dynamic thermal management for internet data centers. Internet Computing, IEEE, 9(1) :42–49, 2005.

[SHUS10] John E Stone, David J Hardy, Ivan S Ufimtsev, and Klaus Schulten. Gpu-accelerated molecular modeling coming of age. Journal of Molecular Graphics and Modelling, 29(2) :116–125, 2010.

[SMLF09] Borja Sotomayor, Rubén S. Montero, Ignacio M. Llorente, and Ian Foster. Virtual infrastructure management in private and hybrid clouds. IEEE Internet Computing, 13(5) :14–22, September 2009.

[SPE08] Specpower altos r380 f2. http://www.spec.org/power_ssj2008/results/res2013q2/power_

ssj2008-20130607-00614.html, 2008. Accessed : 2013-09-01.

[Sta13] Matt Stansberry. The future of green it : Solving the accountability issue. Computer, 46(7) :0091–93, 2013.

[TGV07] Qinghui Tang, S Gupta, and Georgios Varsamopoulos. Thermal-aware task scheduling for data centers through minimizing heat recirculation. In Cluster Computing, 2007 IEEE International Conference on, pages 129–138. IEEE, 2007.

[TYNW07] Kevin Tian, Ke Yu, Jun Nakajima, and Winston Wang. How virtualization makes power management different. In Linux Symposium, page 205, 2007.

[UKIN10] R. Urgaonkar, U.C. Kozat, K. Igarashi, and M.J. Neely. Dynamic resource allocation and power management in virtualized data centers. In Network Operations and Management Symposium (NOMS), 2010 IEEE, pages 479–486, 2010.

[VAN08a] Akshat Verma, Puneet Ahuja, and Anindya Neogi. pmapper : power and migration cost aware application placement in virtualized systems. In Middleware 2008, pages 243–264. Springer, 2008.

[VAN08b] Akshat Verma, Puneet Ahuja, and Anindya Neogi. Power-aware dynamic placement of hpc applications. In Proceedings of the 22nd annual international conference on Supercomputing, pages 175–184. ACM, 2008.

[VBG09] Georgios Varsamopoulos, Ayan Banerjee, and Sandeep K. S. Gupta. Energy efficiency of thermal-aware job scheduling algorithms under various cooling models. In International Conference on Contemporary Computing IC^3, pages 568–580, Noida, India, August 2009.

[WL08] Enhua Wu and Youquan Liu. Emerging technology about gpgpu. In Circuits and Systems, 2008. APCCAS 2008. IEEE Asia Pacific Conference on, pages 618–622. IEEE, 2008.

[WRF+10] Malcolm Ware, Karthick Rajamani, Michael Floyd, Bishop Brock, Juan C Rubio, Freeman Rawson, and John B Carter. Architecting for power management : the ibm® power7™ approach. In High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on, pages 1–11. IEEE, 2010.

[YPZL05] Xudong Yang, Xuexian Pi, Liang Zeng, and Sikun Li. Gpu-based real-time simulation and rendering of unbounded ocean surface. In Computer Aided Design and Computer Graphics, 2005. Ninth International Conference on, pages 6–pp. IEEE, 2005.

Liste des acronymes

API
: Application Programming Interface

AWS
: Amazon Web Service

CFD
: Computational Fluid Dynamics

DVFS
: Dynamic Voltage and Frequency Scaling

EMN
: École des Mines de Nantes

GPGPU
: General Purpose computing on Graphical Processing Unit

HA
: High Availability

HD
: Haute Disponibilité

HPC
: High Performance Computing

IaaS
: Infrastructure as a Service

IPMI

Intelligent Platform Management Interface

JVM

Java Virtual Machine

OS

Operating System

PaaS

Platform as a Service

PPC

Programmation Par Contraintes

REST

REpresentationnal State Transfer

SaaS

Software as a Service

SGIV

Système de Gestion d'Infrastructure Virtualisée

SLA

Service Level Agreement

VM

Virtual Machine

XaaS

X as a Service

Glossaire

Amazon

Amazon.com est une entreprise américaine fournisseur de bien culturel via internet. Étant précurseur du “tout service”, elle propose de plus l’utilisation de ses centres de données en XaaS, et fait partie des 4 plus importantes entreprises en informatiques du monde.

C++

Le C++ est un langage de programmation compilé et orienté objet. Il hérite du lange C, en lui ajoutant entre autres la notion de classes d’objets par tables virtuelles. Le langage C étant très proche de l’assembleur (on par le de *macro-assembleur*), le langage C++ est considéré comme très performant en terme de coût CPU d’exécution. La souplesse de ce langage ajoute cependant des risques d’erreur lors du développement d’un programme, qui de plus doit être compilé pour chaque nouvelle architecture cible.

Choco

Choco est un solveur de contraintes écrit en java et développé à l’EMN

cloud computing

Le modèle de gestion par cloud computing consiste à déporter l’exécution de services informatiques dans un centre dédié. Le client utilisateur des services n’a donc pas connaisance de l’infrastructure réelle du parc, mais agit dans celui-ci au travers des services offerts par le centre. Citons en particulier les modèles SaaS, PaaS et IaaS

Gecode

Gecode, pour GEneric COnstraint Development Environment ou environnement de développement de contraintes générique, est un outil de développement d’applications basées sur la programmation par contraintes. Voir le site `http://www.gecode.org/`

Groovy

Groovy est une extension de Java, et donc un sur-langage de ce dernier, qui autorise en particulier le typage dynamique, implémente des fonctionnalités supplémentaires sur les collections et la gestion des fermetures. Il possède de plus un environnement simple d'exécution de scripts, qui permet d'exécuter des scripts utilisateur.

Java

Java est un langage de programmation oriénté objet. Les programmes sont compilé une première fois en bytecode qui sera interprété par la JVM lors de l'exécution de ce programme.

VMWare

VMware est une entreprise américaine fournissant des SGIV. Elle est principalement connue pour son outil de gestion VCENTER, et ses hyperviseurs dédiés ESX

webserver

Une application de type webserver est caractérisée par une activité CPU fonction de la fréquence d'arrivée de requêtes. Une telle application peut généralement être répartie sur plusieurs serveurs pour réduire la charge de ceux-ci, on parle alors d'application élastique. Les serveurs http, le moteur de recherche de Google sont des exemples d'applications de type webserver.

Xen

Xen est un hyperviseur.

l'université nantes angers le mans
COMMUNAUTÉ D'UNIVERSITÉS ET ÉTABLISSEMENTS

MINES Nantes
Institut Mines-Télécom

Thèse de Doctorat

Guillaume LE LOUËT

Maîtrise énergétique des centres de données virtualisés

D'un scénario de charge à l'optimisation du placement des calculs

Power management in virtualized data centers

From a load scenario to the optimization of the tasks placement

Résumé
Cette thèse se place dans le contexte de l'hébergement de services informatiques virtualisés et apporte deux contributions.
Elle propose premièrerent un système d'aide à la gestion modulaire, déplaçant les machines virtuelles du centre pour le maintenir dans un état satisfaisant. Ce système permet en particulier d'intégrer la notion de consommation électrique des serveurs ainsi que des règles propres à cette consommation. Sa modularité permet de plus l'adaptation de ses composants à des problèmes de grande taille.
Cette thèse propose de plus un outil pour comparer différents gestionnaires de centres virtualisés. Cet outil injecte un scénario de montée en charge reproductible dans une infrastructure virtualisée. L'injection d'un tel scénario permet d'évaluer les performances du système de gestion du centre grâce à des sondes spécifiques. Le langage utilisé pour cette injection est extensible et permet l'utilisation de scénarios paramétrés.

Mots clés
placement des machines virtuelles, gestionnaire de centre de données, programmation par contrainte, injection de charge, scénario d'activités, performances des systèmes virtualisés

Abstract
This thesis considers the virtualized IT services hosting and makes two contributions.
It first proposes a modular system of management aids, to move the virtual machines of the center in order to keep it in a good condition. This system allows in particular to integrate the concept of server power consumption and rules specific to that concept. What's more, its modularity allows to adjust its components to handle larger problems.
This thesis proposes also a tool to compare different virtualized centers managers. This tool injects a reproductible load increase scenario in a virtualized infrastructure. The injection of such a scenario is used to evaluate the performance of the system center manager, using performances probes. The language used for this injection is extensible and allows the creation of parameterized scenarios.
The contributions of this thesis were presented in two international conferences and a french conference.

Key Words
virtual machine placement, data center manager, constraint programing, load injection, activity scenario, performances of virtualized systems